



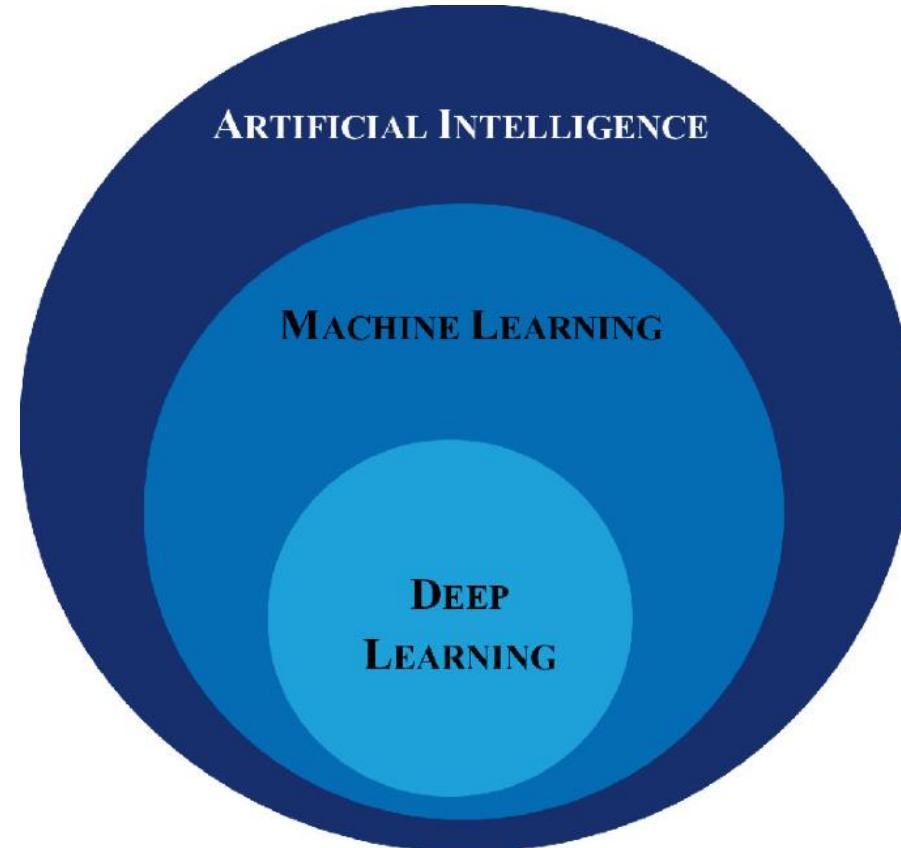
دوره آموزش مهارت محور اینترنت اشیا
تیر و مرداد ۱۴۰۳

سینا مجیدی

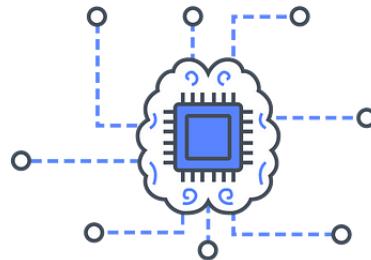
کارشناس اینترنت اشیا

بخش AIoT

هوش مصنوعی، یادگیری ماشین و یادگیری عمیق



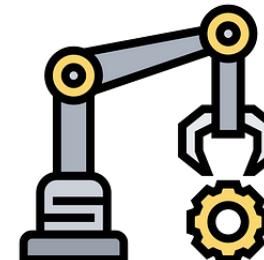
شاخه های هوش مصنوعی



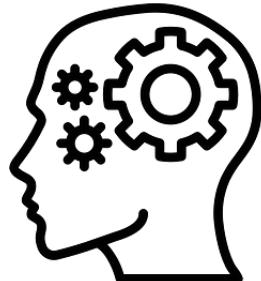
Machine Learning



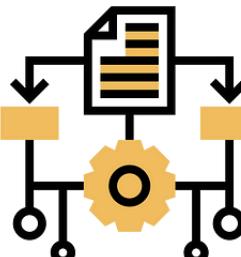
Neural Networks



Robotics



Expert Systems



Fuzzy Logic

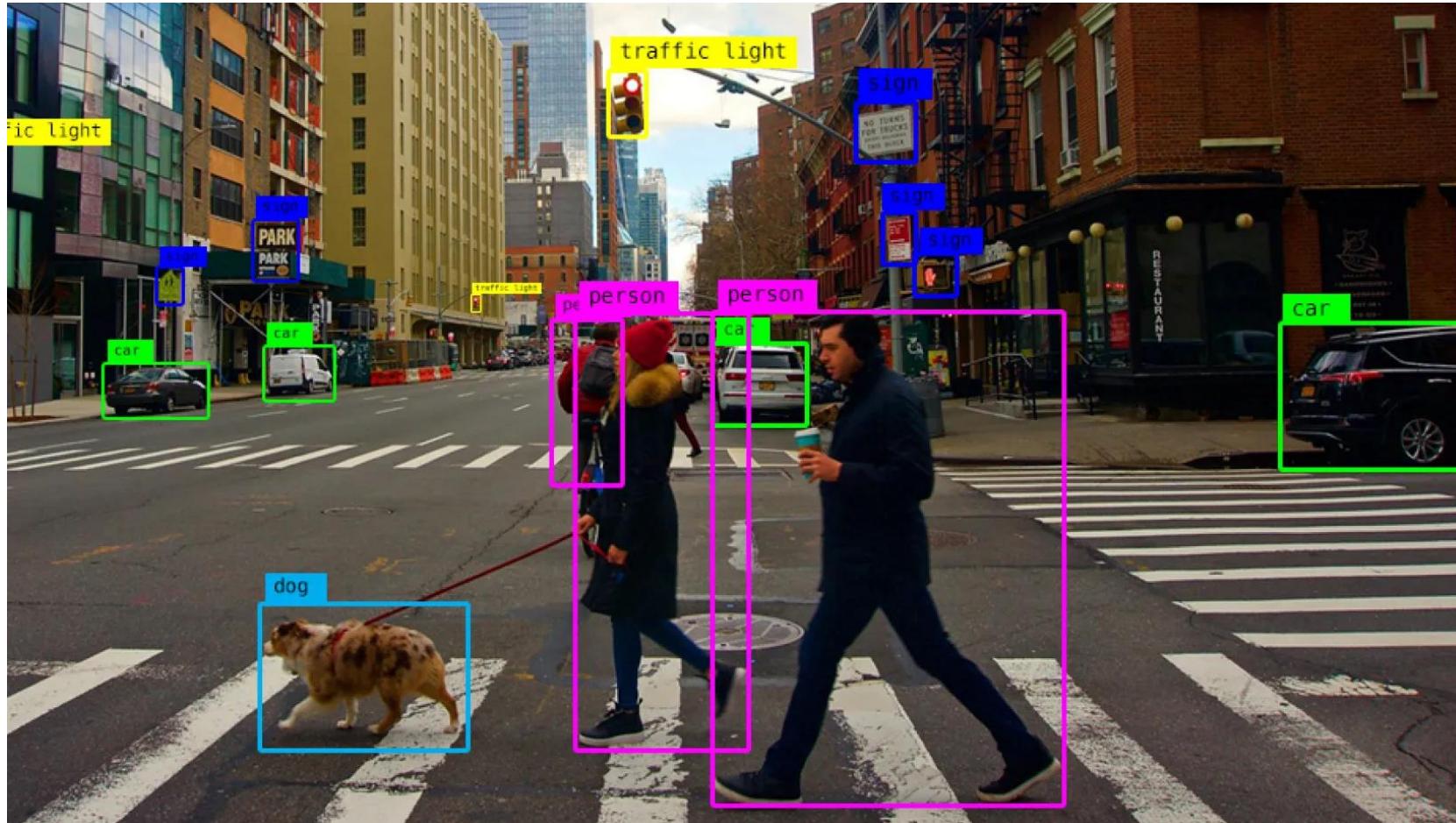


Natural Language Processing

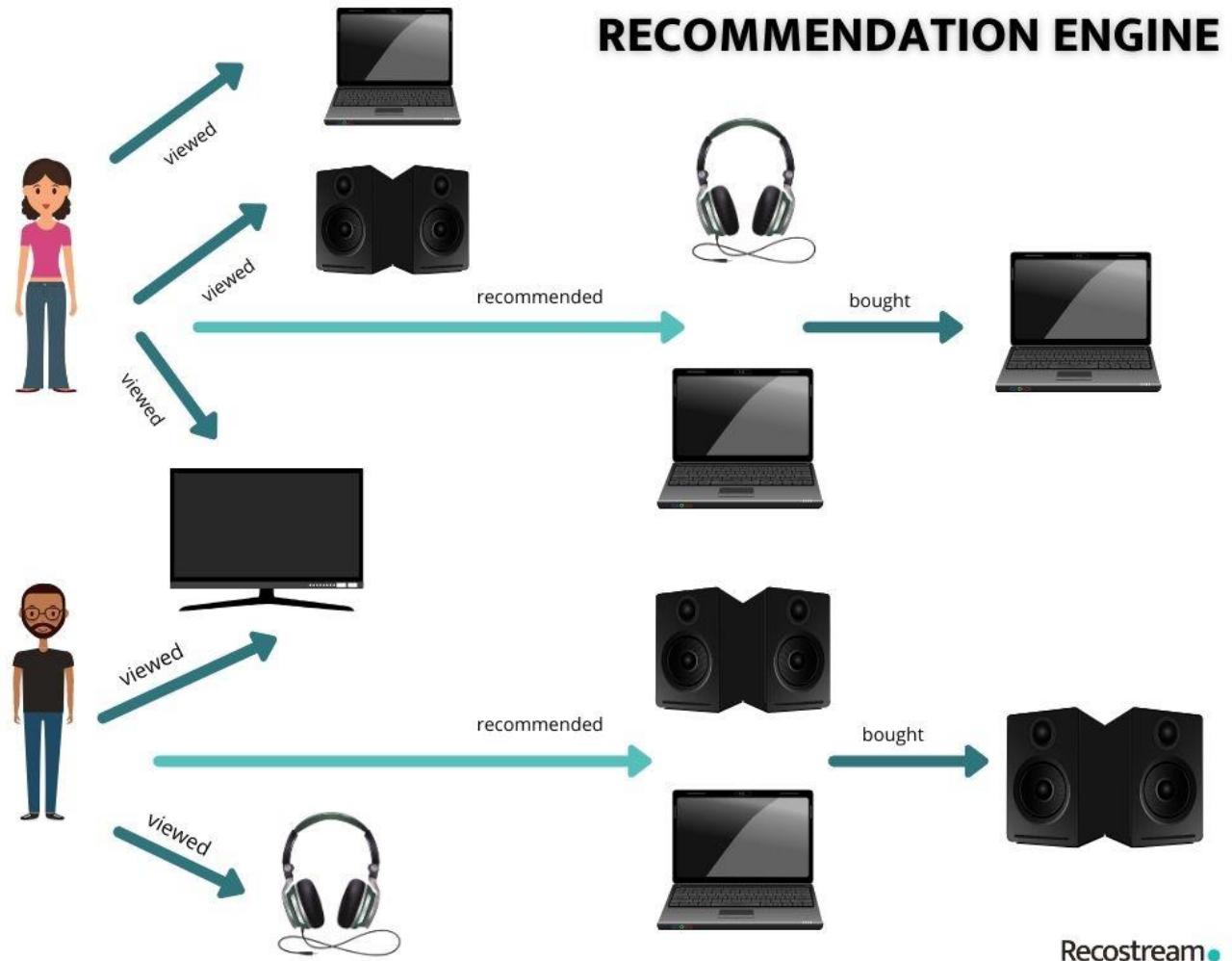
کاربردهای هوش مصنوعی (NLP)



کاربرد های هوش مصنوعی (Object detection)



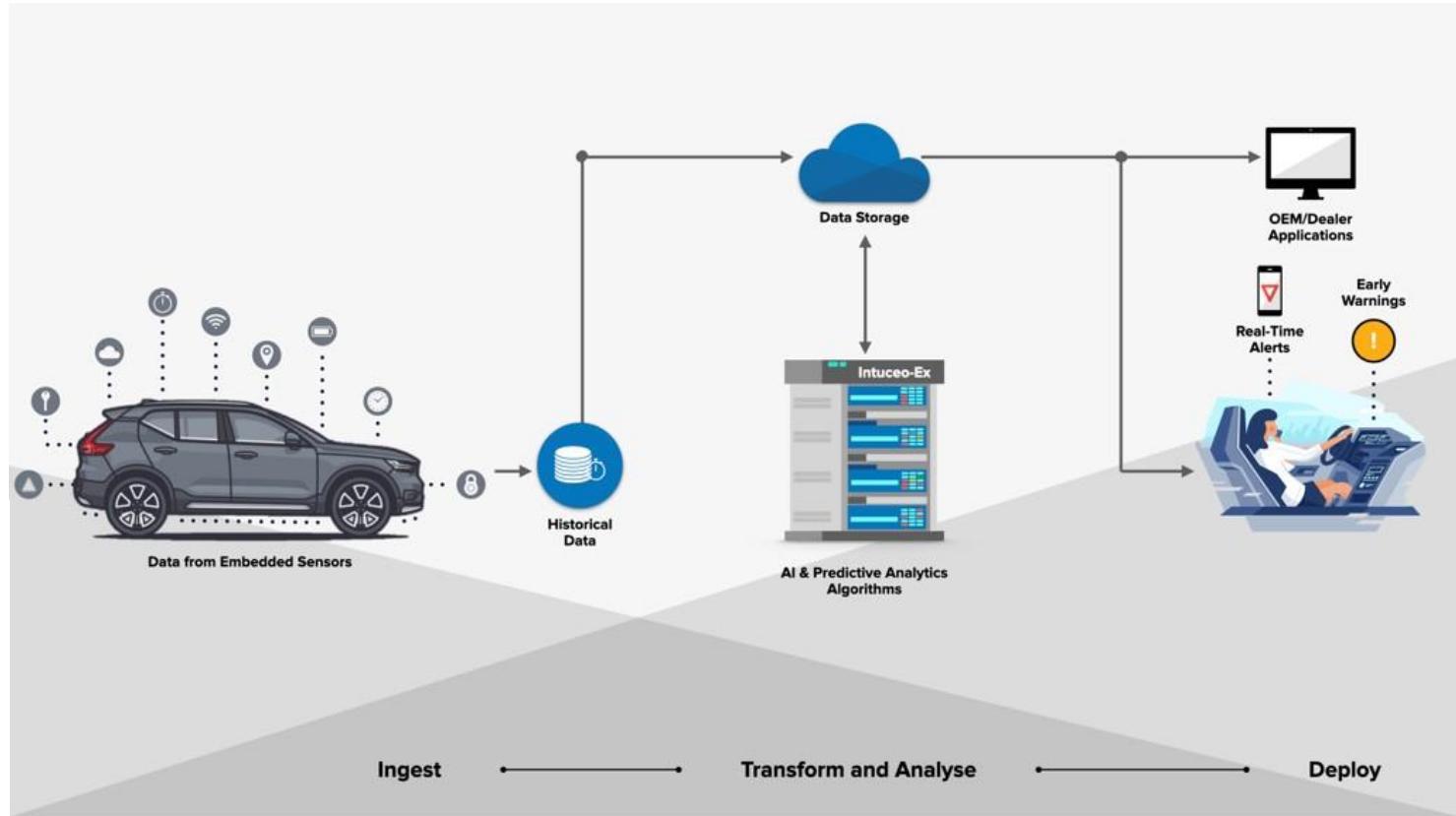
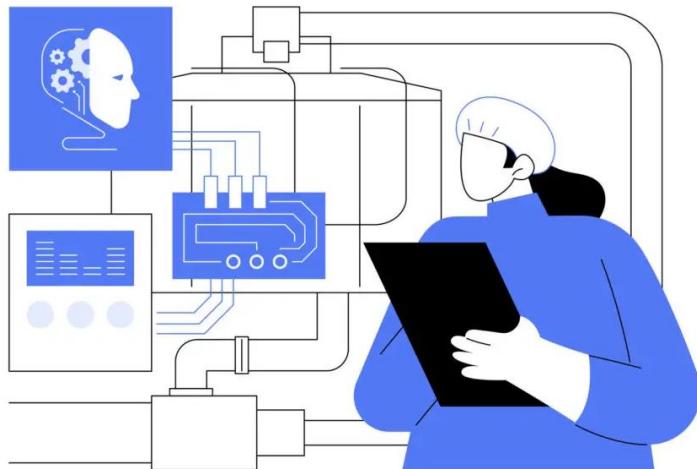
کاربرد های هوش مصنوعی (recommender systems)



کاربردهای هوش مصنوعی (Self drive cars and transportation)

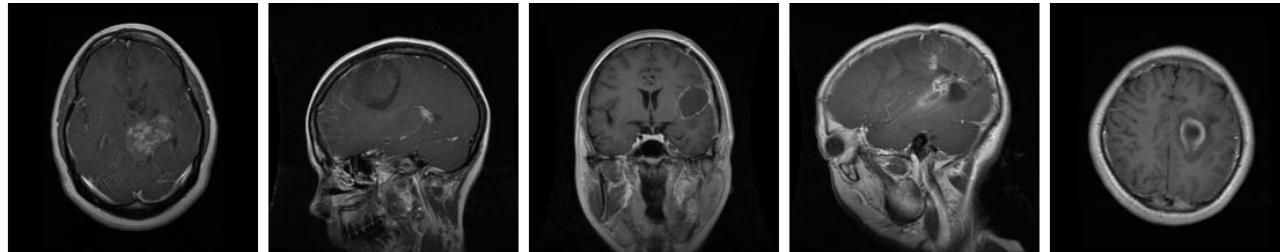


کاربردهای هوش مصنوعی (predictive maintenance)

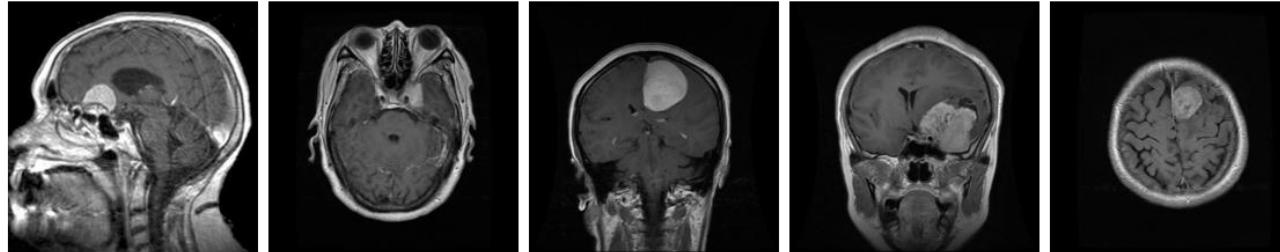


کاربردهای هوش مصنوعی (AI disease prediction)

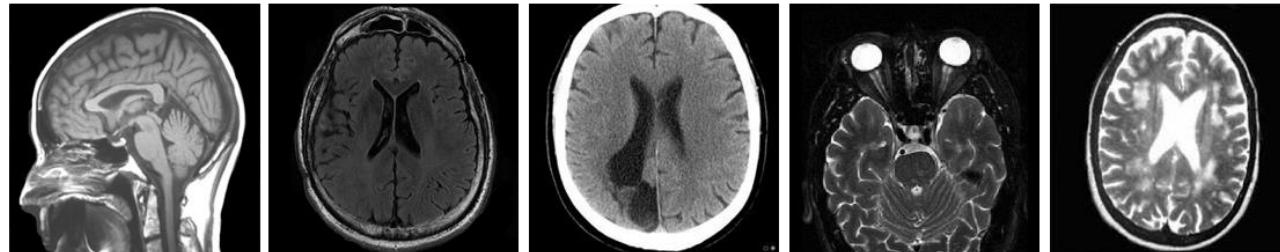
Glioma



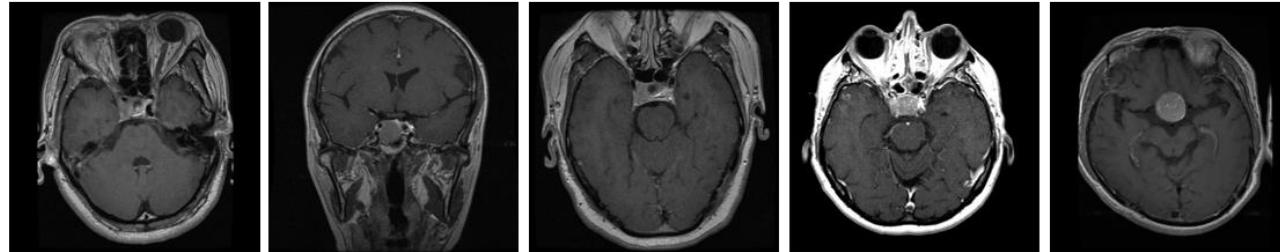
Meningioma



No-Tumor



Pituitary



IOT layers

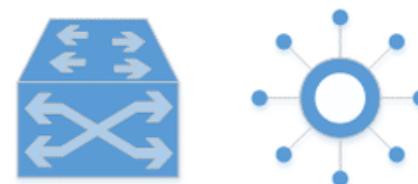
Application Layer



Data Processing Layer



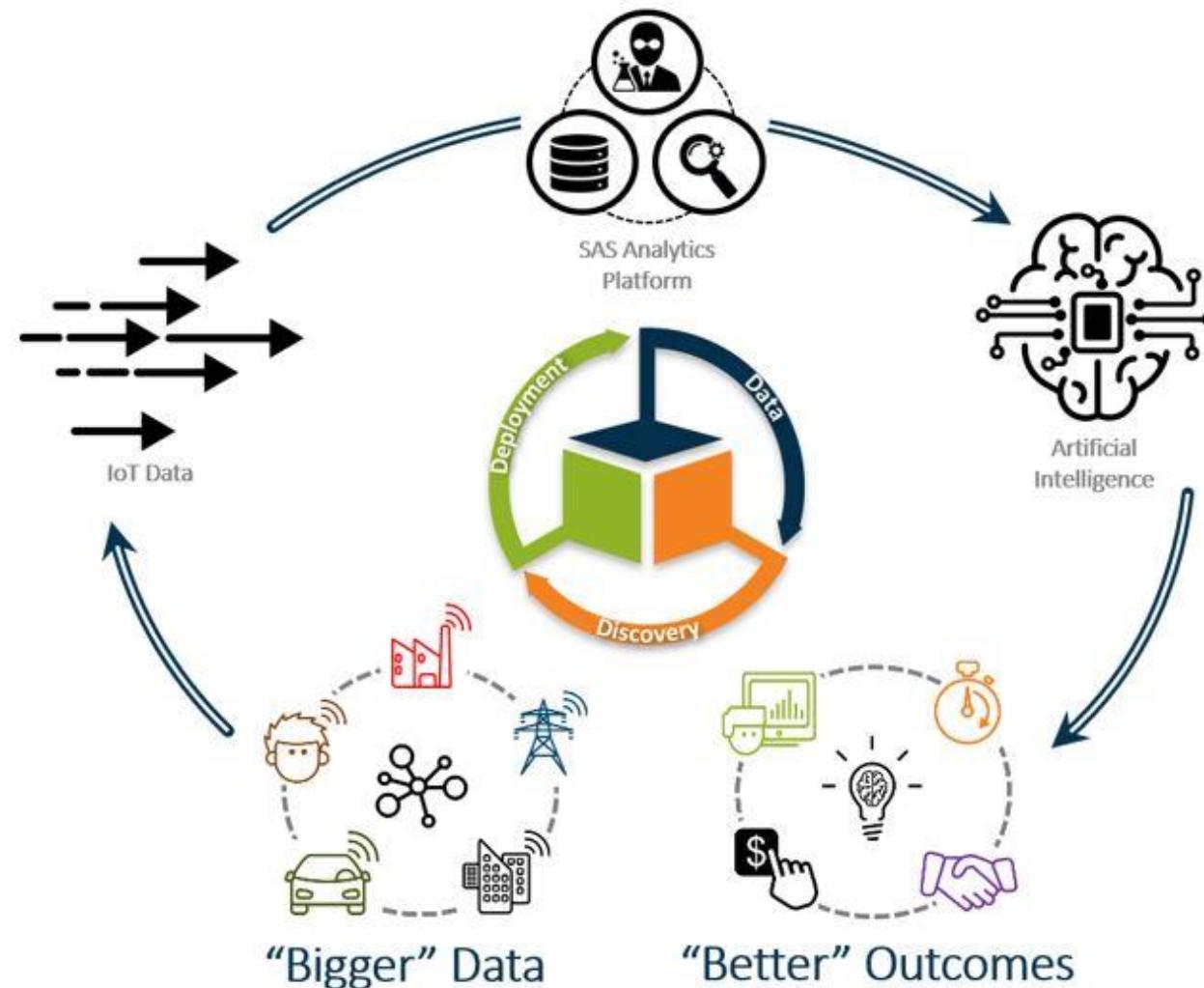
Networking Layer



Sensors and Actuators Layer



AIOT

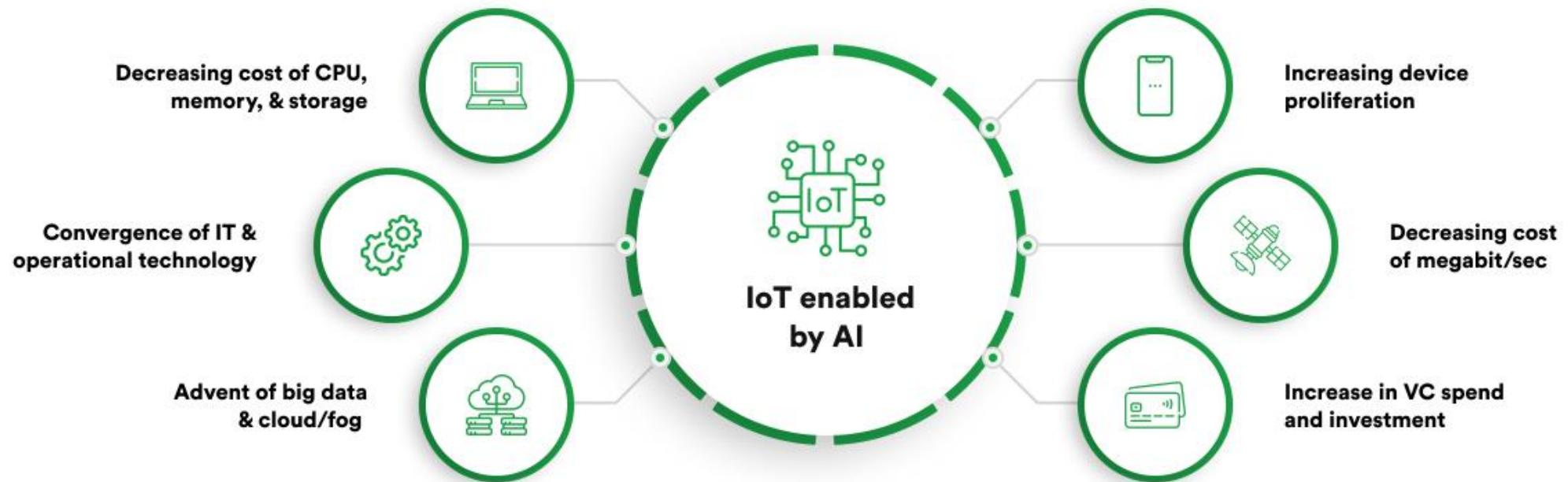


AIoT

Role of AIoT
In Smart Traffic
Solutions



AIOT



Edge computing

explain edge computing like im 5 years old



Okay! Imagine you have a toy robot that you love playing with. Sometimes, you ask the robot to do something, like dance or sing. Now, let's pretend your robot has to call someone far away every time you ask it to dance, and that person tells the robot what to do. That would take a long time, right?

But what if the robot could figure out how to dance all by itself, without calling anyone? It would be much faster! That's kind of what edge computing does.

Instead of sending information to a faraway place (like the cloud) to get answers, edge computing lets devices (like your robot or phone) do the thinking close by, right where they are. This makes things happen quicker!



Edge computing

Imagine you have a toy robot that you like to play with. Sometimes, you want the bot to do something, like dance or sing. Now, let's pretend that your robot needs to call someone far away every time you ask it to dance, and that person tells the robot what to do. This takes a long time, right?

But what if the robot could figure out how to dance on its own without contacting anyone? It would be much faster! This is what edge computing does.

Instead of sending information to a distant location (like the cloud) to get a response, edge computing allows devices (like your robot or phone) to think right where they are. It makes everything happen faster!



تصور کنید یک ربات اسباب بازی دارید که دوست دارد با آن بازی کنید. گاهی اوقات، شما می خواهید ربات کاری انجام دهد، مانند رقص با آواز خواندن. حالا، باید وانمود کنیم که ربات شما باید هر بار که از او می خواهید برقصد، با شخصی دور تماس بگیرد، و آن شخص به ربات می گوید که چه کاری انجام دهد. این زمان زیادی طول می کشد، درست است؟

اما اگر ربات بتواند به تنهایی و بدون تماس با کسی بفهمد چگونه می رقصد چه؟ خیلی سریعتر می شد! این کاری است که رایانش لبه انجام می دهد.

به جای ارسال اطلاعات به یک مکان دور (مانند ابر) برای دریافت پاسخ، محاسبات لبه به مستگاهها (مانند ربات یا تلفن شما) اجازه می دهد دقیقاً در کجا هستند فکر کنند. این باعث می شود همه چیز سریعتر اتفاق بیفتد!

tasor konid yek rabat asbab bazi darid keh dost darid ba an bazi konid. gangi oqat, shma mi khaehid rabat kari enjam dahad, manand raqs ya avaz khandan. hala, biaeed vanemud konim keh rabat shma bayad npar bar keh az o mikhaehid barghsad, ba shkhs Show more

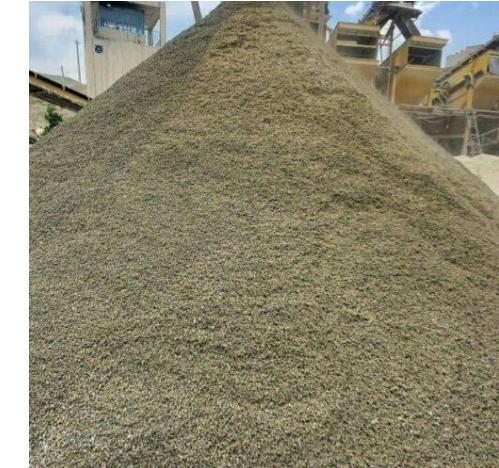
Edge computing



processing and analysis

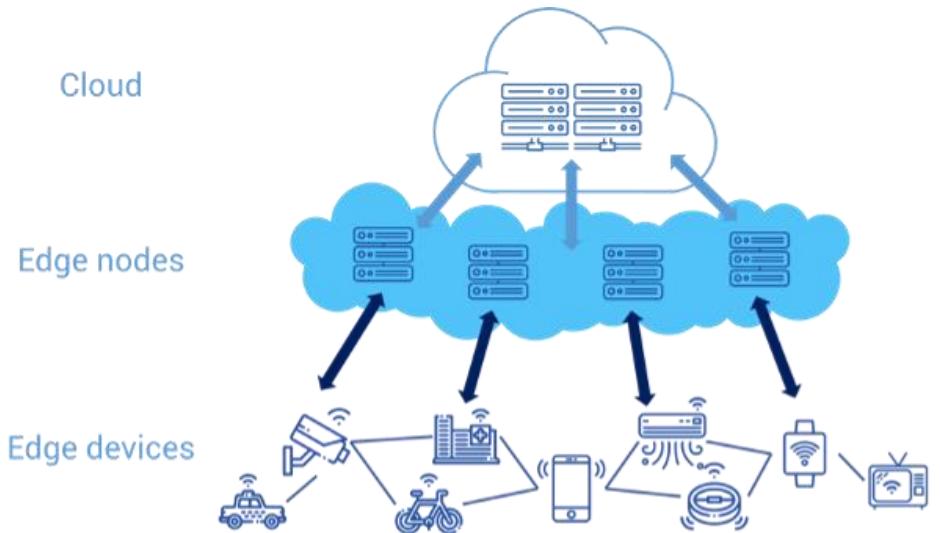
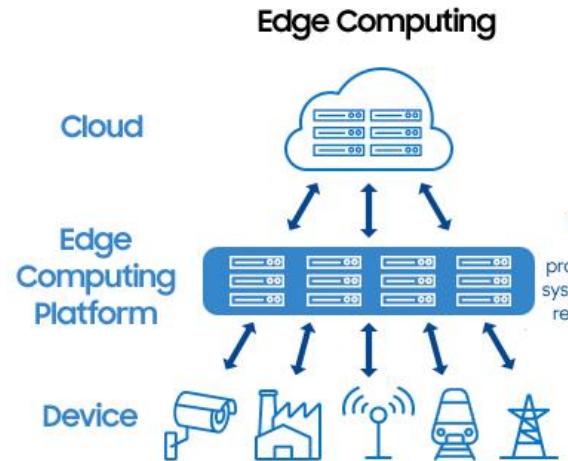
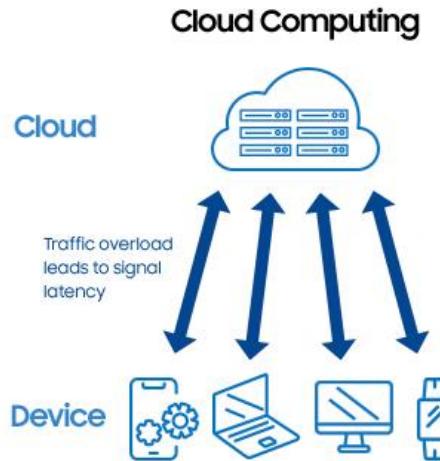


Edge computing

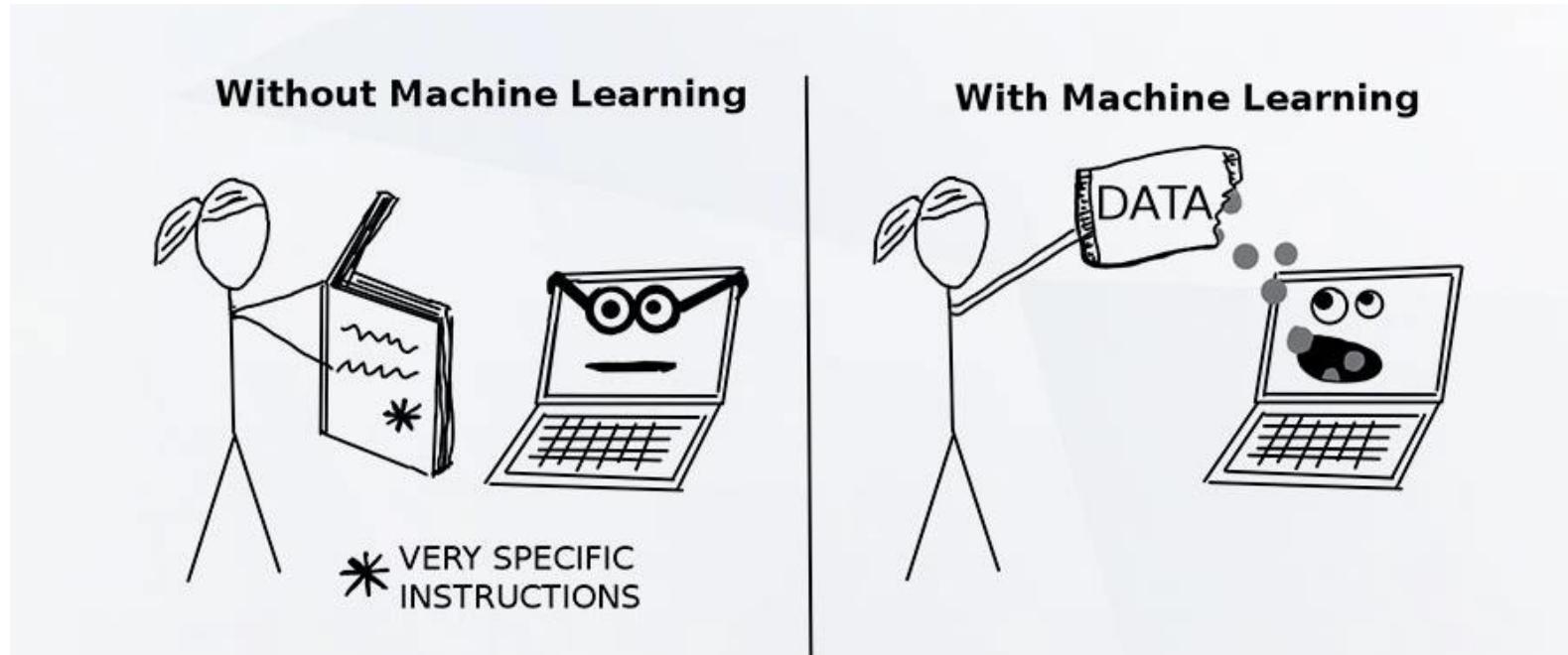


data

Edge computing

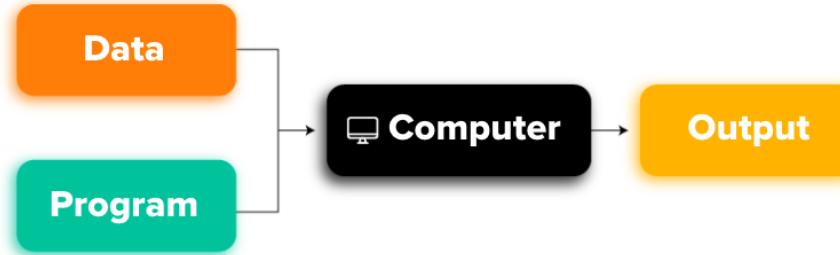


سیستم سنتی VS یادگیری ماشین

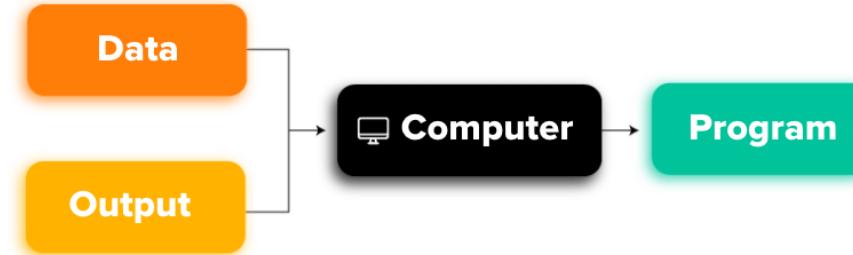


سیستم سنتی VS یادگیری ماشین

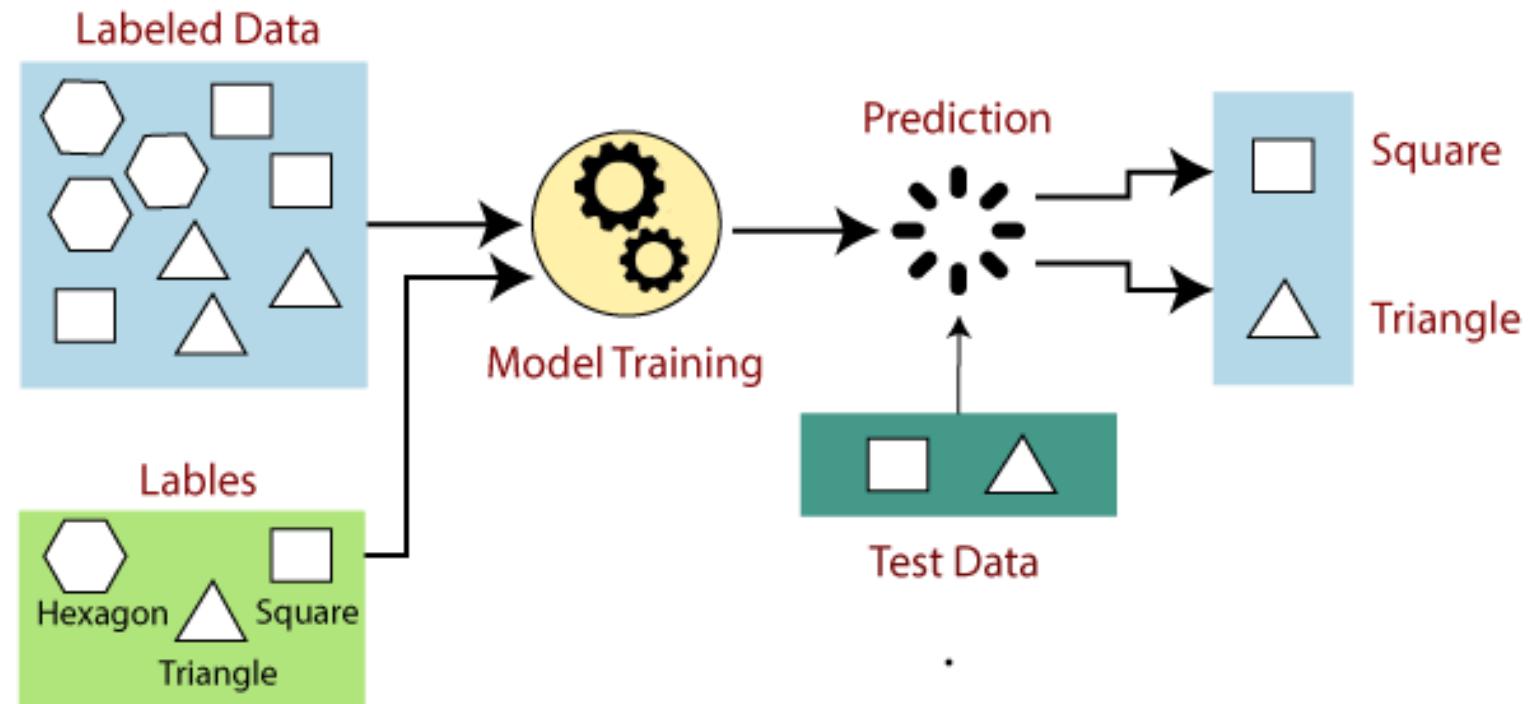
TRADITIONAL PROGRAMMING



MACHINE LEARNING

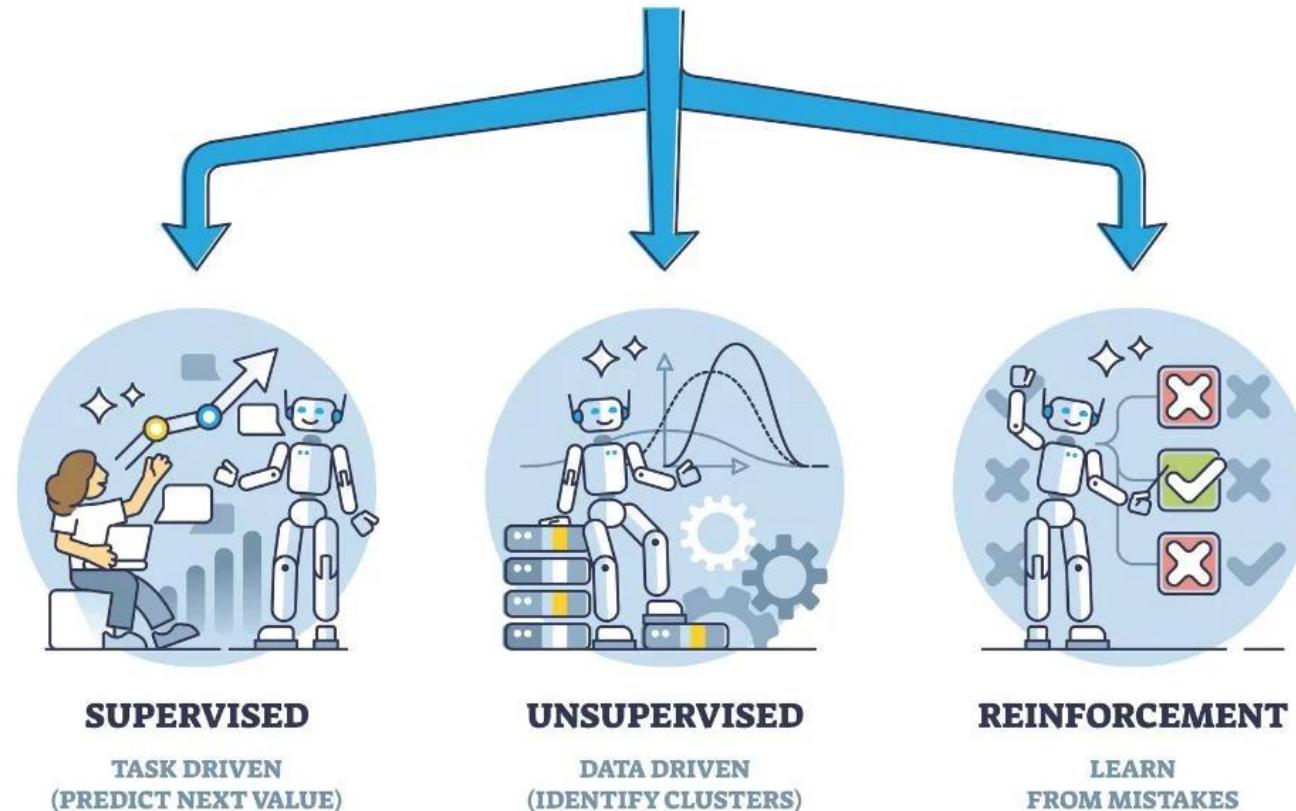


یادگیری ماشین

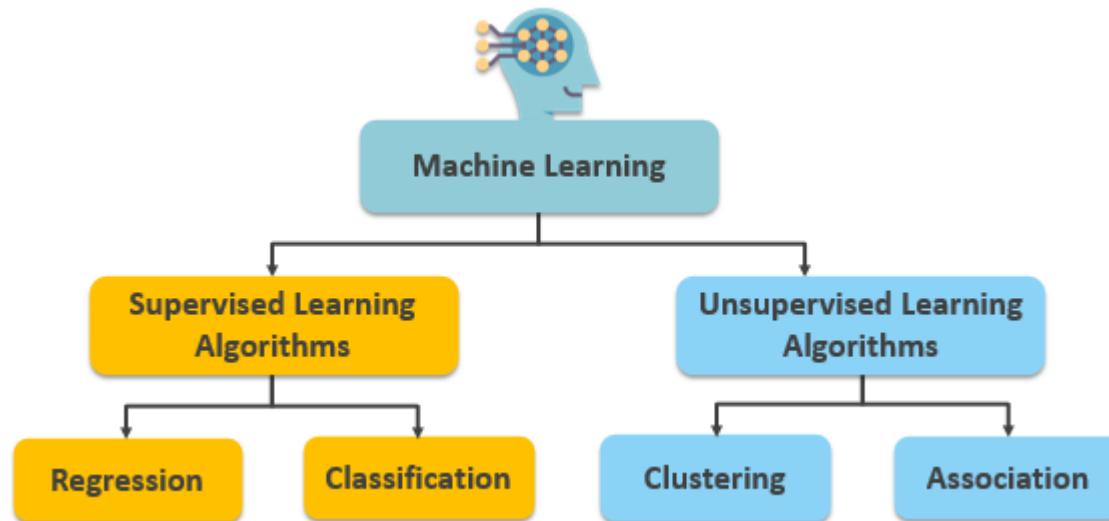


یادگیری ماشین (ناظرت شده و بی ناظرت)

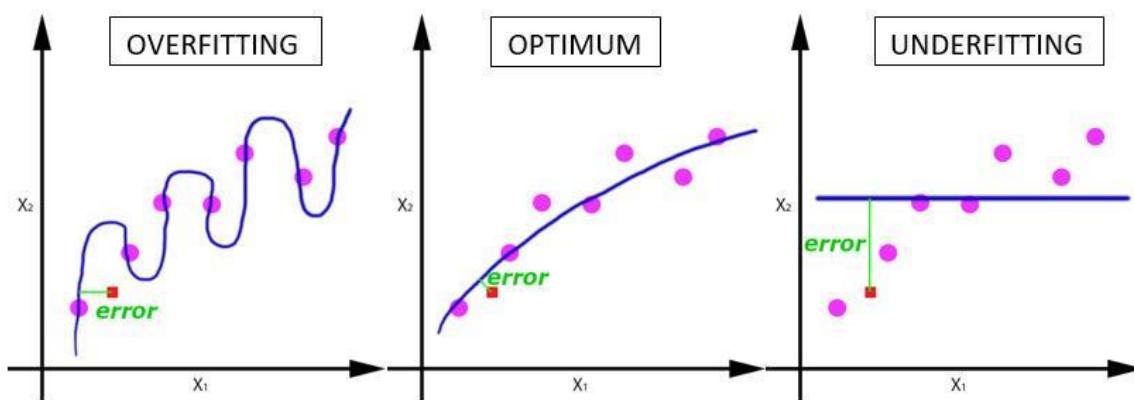
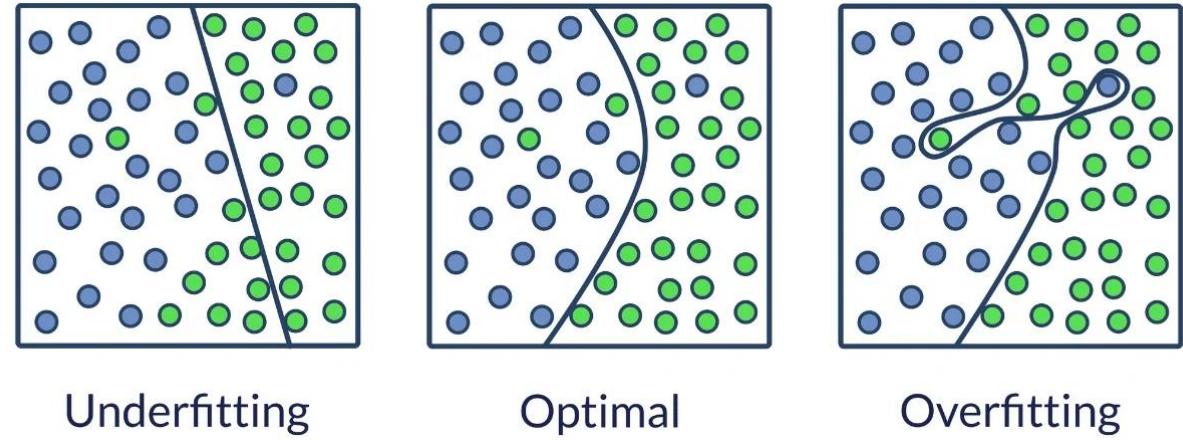
MACHINE LEARNING



الگوریتم های یادگیری ماشین (ناظارت شده و بی ناظرت)



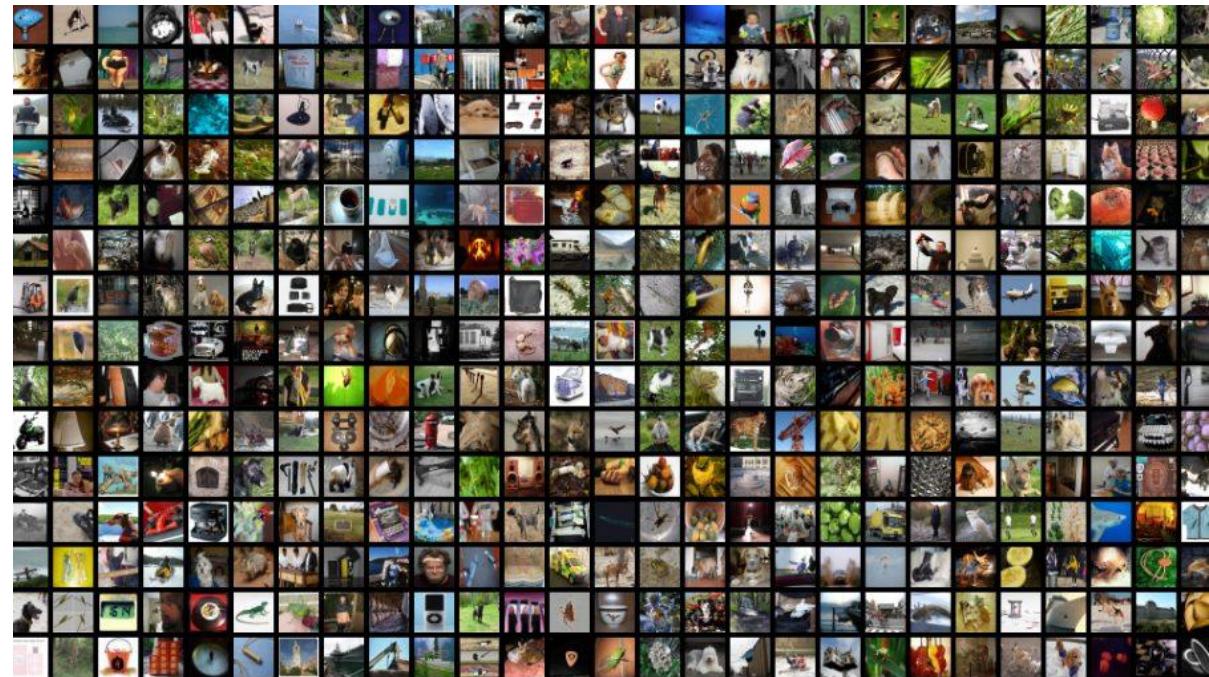
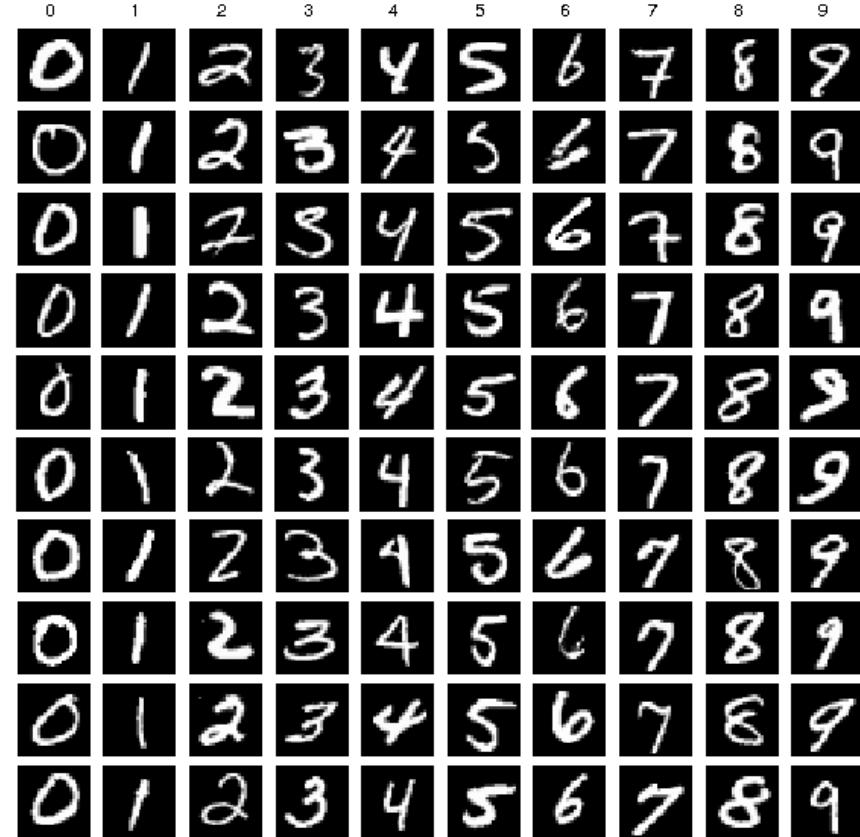
Overfitting and Underfitting



dataset

| Car | Model | Volume | Weight | CO2 |
|------------|------------|--------|--------|-----|
| Toyota | Aygo | 1000 | 790 | 99 |
| Mitsubishi | Space Star | 1200 | 1160 | 95 |
| Skoda | Citigo | 1000 | 929 | 95 |
| Fiat | 500 | 900 | 865 | 90 |
| Mini | Cooper | 1500 | 1140 | 105 |
| VW | Up! | 1000 | 929 | 105 |
| Skoda | Fabia | 1400 | 1109 | 90 |
| Mercedes | A-Class | 1500 | 1365 | 92 |
| Ford | Fiesta | 1500 | 1112 | 98 |
| Audi | A1 | 1600 | 1150 | 99 |
| ... | ... | ... | ... | ... |

dataset



confusion matrix

| | | True Class | |
|-----------------|----------|------------|----------|
| | | Positive | Negative |
| Predicted Class | Positive | TP | FP |
| | Negative | FN | TN |

Accuracy

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$



Recall

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$



Precision

Precision = $\frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$



Some math prerequisites

1. Mean (Average):

The **mean** is the average of a set of numbers. It is calculated by summing all the values in the dataset and dividing by the number of values.

Formula:

$$\text{Mean} = \frac{\sum_{i=1}^n x_i}{n}$$

Where x_i represents each data point, and n is the total number of data points.

Example:

Consider the dataset: [10, 20, 30, 40, 50]

- Sum of the values: $10 + 20 + 30 + 40 + 50 = 150$
- Number of values: 5



Some math prerequisites

2. Standard Deviation (std):

The **standard deviation** is a measure of the spread or dispersion of a set of data points from their mean. A low standard deviation means the data points are close to the mean, while a high standard deviation means the data points are more spread out.

Formula:

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^n (x_i - \text{mean})^2}{n}}$$

Example:

Using the dataset: [10, 20, 30, 40, 50], and the mean (30) from the previous example:

- $(10 - 30)^2 = 400$
- $(20 - 30)^2 = 100$
- $(30 - 30)^2 = 0$
- $(40 - 30)^2 = 100$
- $(50 - 30)^2 = 400$

Sum of squared differences:

$$400 + 100 + 0 + 100 + 400 = 1000$$

Some math prerequisites

Dividing by the number of values:

$$\frac{1000}{5} = 200$$

Finally, taking the square root gives the **standard deviation**:

$$\sqrt{200} \approx 14.14$$

So, the standard deviation of the dataset is approximately **14.14**.

Some math prerequisites

3. Mean Squared Error (MSE):

The **mean squared error (MSE)** is a measure used in regression analysis and machine learning to quantify how far predicted values are from the actual values. It calculates the average of the squares of the differences (errors) between predicted values and actual values.

Formula:

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

Where:

- y_i is the actual value,
- \hat{y}_i is the predicted value,
- n is the number of data points.

Example:

Suppose we have the actual values: [10, 20, 30] and predicted values: [12, 18, 33]

Step 1: Calculate the squared differences between the actual and predicted values:

- $(10 - 12)^2 = 4$
- $(20 - 18)^2 = 4$
- $(30 - 33)^2 = 9$

Some math prerequisites

Step 2: Sum the squared differences:

$$4 + 4 + 9 = 17$$

Step 3: Divide by the number of data points (3 in this case):

$$\frac{17}{3} \approx 5.67$$

So, the **MSE** is approximately 5.67.

numpy

```
import numpy as np

# Original array
a = np.array(['1', '2', '3', '4'])
print(a, " ", a.shape)

# Reshaping to (4, 1)
reshaped_a = a.reshape((4, 1))
print(reshaped_a, " ", reshaped_a.shape)

# Reshaping using -1 for unknown dimension
reshaped_a = a.reshape((2, -1))
print(reshaped_a, " ", reshaped_a.shape)
```



numpy

```
import numpy as np

# Creating a vector (1D array)
vector = np.array([1, 2, 3, 4, 5])
print("Vector:\n", vector)

# Creating a matrix (2D array)
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Matrix:\n", matrix)

# Creating a tensor (3D array)
tensor = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("Tensor:\n", tensor)
```

numpy

```
import numpy as np

# Creating vectors (1D arrays)
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# Vector operations
print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b) # Element-wise multiplication
print("a + 5 =", a + 5) # Data broadcasting: add 5 to each element of the array
```

numpy

```
import numpy as np

a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Slicing the array from index 2 to 4 (5 is excluded)
print("a[2:5] =", a[2:5])

# Filtering elements greater than 3
print("a[a > 3] =", a[a > 3])
```

numpy

```
import numpy as np

# Create a numpy array
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Perform numpy operations and print results
print("Minimum value:", np.min(a))      # Minimum value in the array
print("Maximum value:", np.max(a))      # Maximum value in the array
print("Sum of elements:", np.sum(a))     # Sum of all elements in the array
print("Average value:", np.average(a))   # Average of all elements in the array
print("Sorted array:", np.sort(a))       # Sorted version of the array
```

numpy

```
import numpy as np

# Create a numpy array
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Perform numpy operations and print results
print("Minimum value:", np.min(a))      # Minimum value in the array
print("Maximum value:", np.max(a))      # Maximum value in the array
print("Sum of elements:", np.sum(a))     # Sum of all elements in the array
print("Average value:", np.average(a))   # Average of all elements in the array
print("Sorted array:", np.sort(a))       # Sorted version of the array
```

numpy

```
import numpy as np
```

Generate a 2x2 array of random integers between 0 and 100

```
random_np_array = np.random.randint(0, 100, (2, 2))
```

Print the generated random array

```
print(random_np_array)
```

pandas

```
import pandas as pd

# Data
data = {
    "temp": [35, 33, 31, 29, 23, 27, 34, 35, 19, 3],
    "humidity": [45, 55, 61, 32, 56, 78, 24, 56, 65, 44]
}

# Create DataFrame
df = pd.DataFrame(data)

# Export DataFrame to CSV file
df.to_csv('output.csv', index=False)

# Print the DataFrame
print(df)
```



pandas

```
import pandas as pd

# Read the CSV file
read_csv = pd.read_csv('output.csv')

# Print the entire DataFrame
print(read_csv)

# Display the first few rows of the DataFrame
print(read_csv.head())

# Get descriptive statistics of the DataFrame
print(read_csv.describe())

# Print the 'temp' column from the original DataFrame (if df is defined)
# Assuming df was defined earlier, else you should use read_csv['temp'] instead
print(read_csv['temp']) # Updated to read from read_csv

# Print the row at index 2
print(read_csv.iloc[2])

# Print the value at row index 2 and column index 1
print(read_csv.iloc[2, 1])
```

pandas

```
import pandas as pd

# Read CSV file
read_csv = pd.read_csv('output.csv')
print("Data from CSV:\n", read_csv)

# Filter rows where temperature is greater than 30
filtered_df = read_csv[read_csv['temp'] > 30]
print("\nFiltered Data (temp > 30):\n", filtered_df)

# Sort the DataFrame by 'temp' in descending order
sorted_df = read_csv.sort_values('temp', ascending=False)
print("\nSorted Data (by temp descending):\n", sorted_df)

# Group the DataFrame by 'temp' and count the occurrences
df_grouped = read_csv.groupby('temp')
print("\nGrouped Data (count of each temp):\n", df_grouped.count())
```

pandas

```
import pandas as pd

# Read the CSV file
read_csv = pd.read_csv('output.csv')

# Add a new column 'wind_speed'
read_csv['wind_speed'] = [1, 2.3, 4, 3.3, 5, 2, 1.5, 2.7, 3.2, 4] # Make sure the list is same length as rows
print("DataFrame after adding 'wind_speed':\n", read_csv)

# Drop the 'wind_speed' column
read_csv = read_csv.drop('wind_speed', axis=1)
print("\nDataFrame after dropping 'wind_speed':\n", read_csv)

# Append a new row to the DataFrame
read_csv.loc[len(read_csv)] = {
    "temp": 33,
    "humidity": 18
}
print("\nDataFrame after appending a new row:\n", read_csv)
```

matplotlib

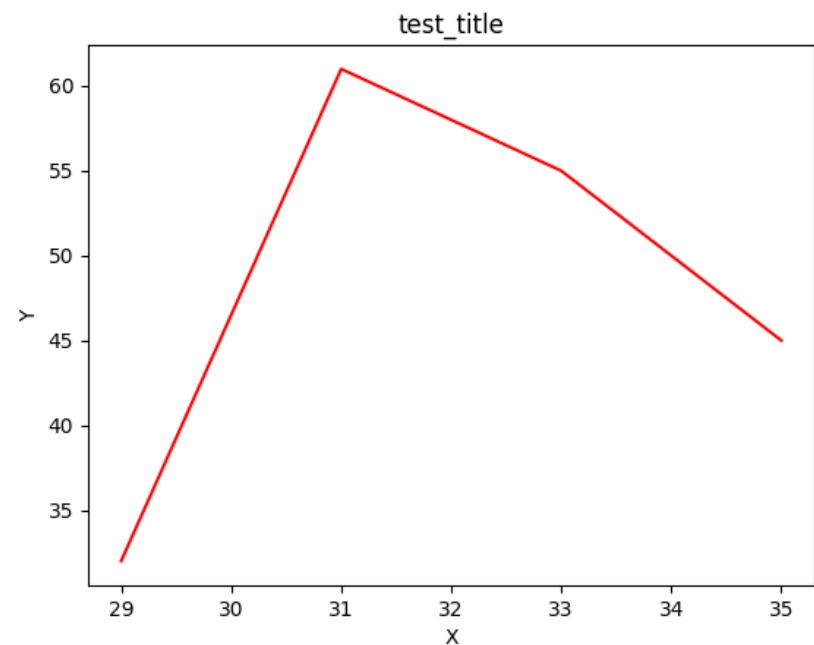
```
import matplotlib.pyplot as plt

# Data for the plot
x = [35, 33, 31, 29]
y = [45, 55, 61, 32]

# Create a simple line plot
plt.plot(x, y, color='red') # Red colored line for the plot

# Add a title and labels for the axes
plt.title("test_title") # Set the plot title
plt.xlabel("X")          # Label for the X-axis
plt.ylabel("Y")          # Label for the Y-axis

# Display the plot
plt.show()
```



matplotlib

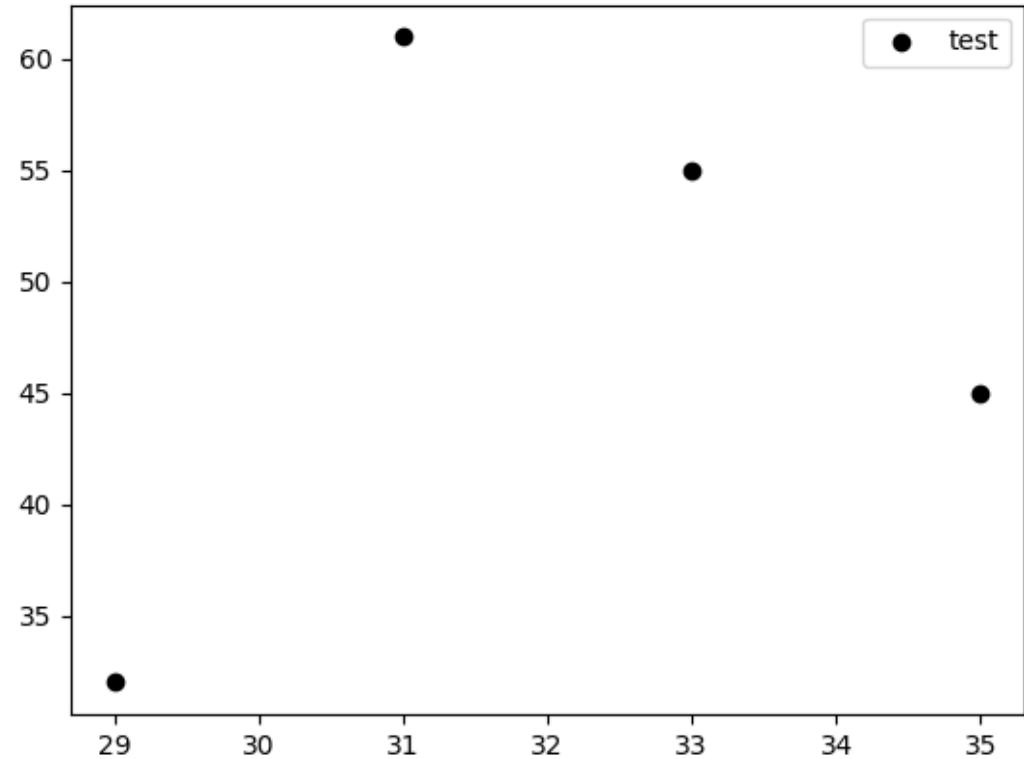
```
import matplotlib.pyplot as plt

x = [35, 33, 31, 29]
y = [45, 55, 61, 32]

# Create a scatter plot
plt.scatter(x, y, color="black", label="test")

# Add a Legend to the plot
plt.legend()

# Show the plot
plt.show()
```



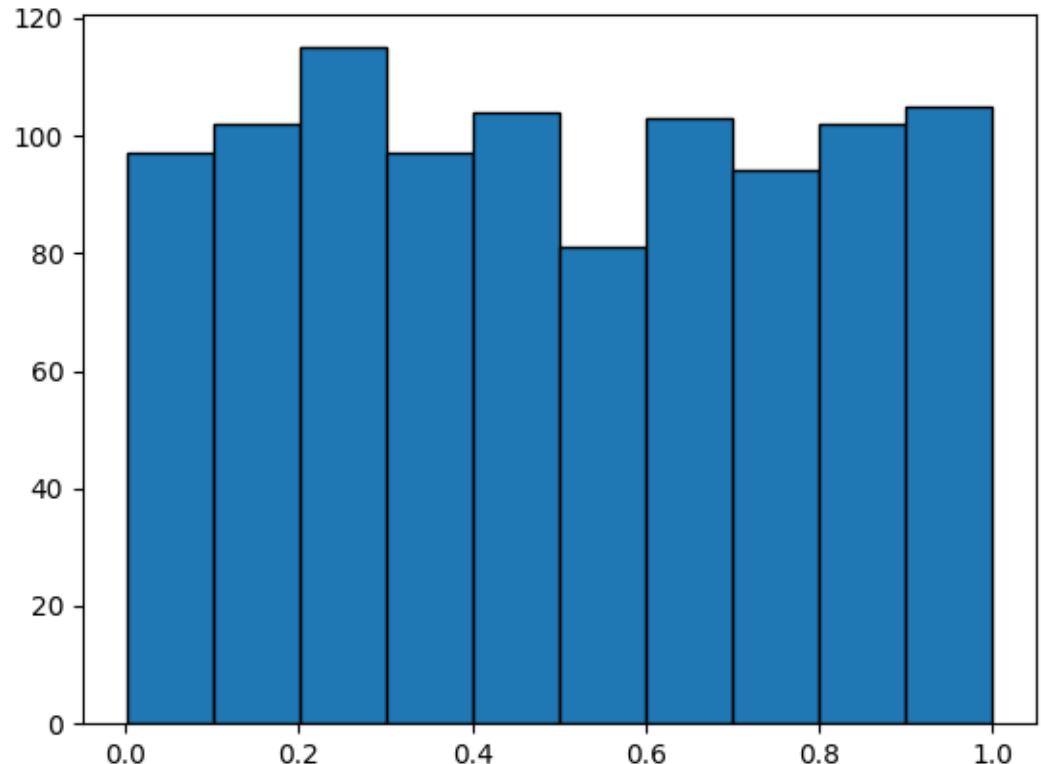
matplotlib

```
import matplotlib.pyplot as plt
import numpy as np

# Generate 1000 random numbers between 0 and 1
x = np.random.rand(1000, 1)

# Plot a histogram with 10 bins
plt.hist(x, bins=10, edgecolor='black')

# Display the plot
plt.show()
```



Scaling and normalizing

| Car | Model | Volume | Weight | CO2 |
|------------|------------|--------|--------|-----|
| Toyota | Aygo | 1.0 | 790 | 99 |
| Mitsubishi | Space Star | 1.2 | 1160 | 95 |
| Skoda | Citigo | 1.0 | 929 | 95 |
| Fiat | 500 | 0.9 | 865 | 90 |
| Mini | Cooper | 1.5 | 1140 | 105 |
| VW | Up! | 1.0 | 929 | 105 |
| Skoda | Fabia | 1.4 | 1109 | 90 |
| Mercedes | A-Class | 1.5 | 1365 | 92 |
| Ford | Fiesta | 1.5 | 1112 | 98 |
| Audi | A1 | 1.6 | 1150 | 99 |
| ... | ... | ... | ... | ... |

Scaling and normalizing

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

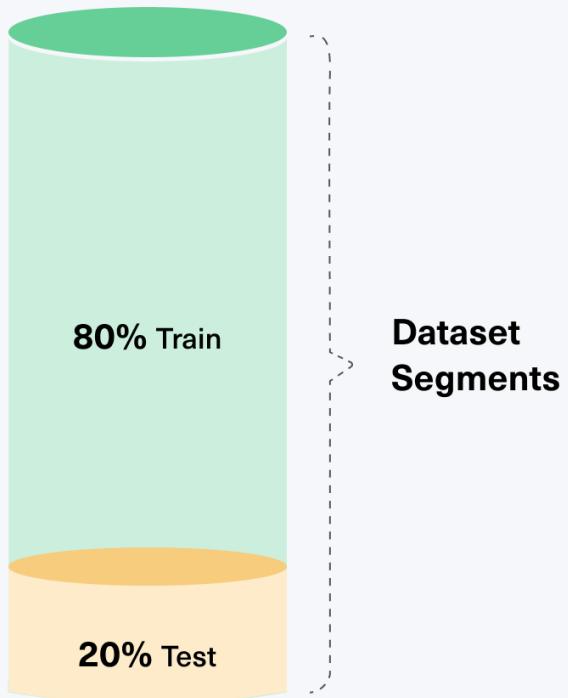
df = pd.read_csv('heart_statlog_cleveland_hungary_final.csv')
print(df.head())
print(df.shape)
print(df['target'].value_counts())

x = df.drop('target', axis=1)
y = df['target']
print(x.shape, y.shape)

x = np.array(x)
y = np.array(y)
print(x.shape, y.shape)

scaler = StandardScaler()
x = scaler.fit_transform(x)
```

train test split



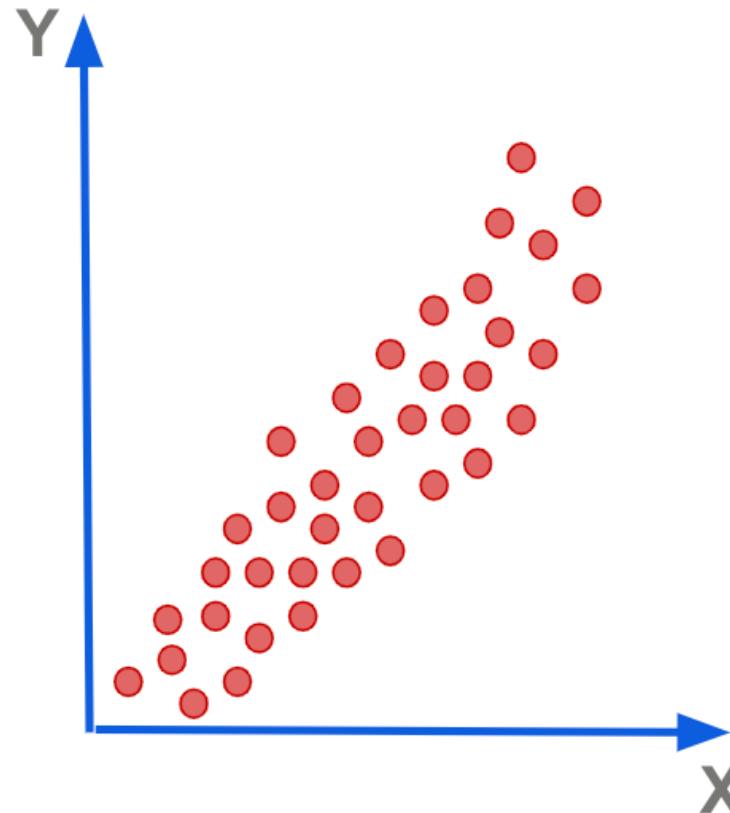
train test split

```
from sklearn.model_selection import train_test_split

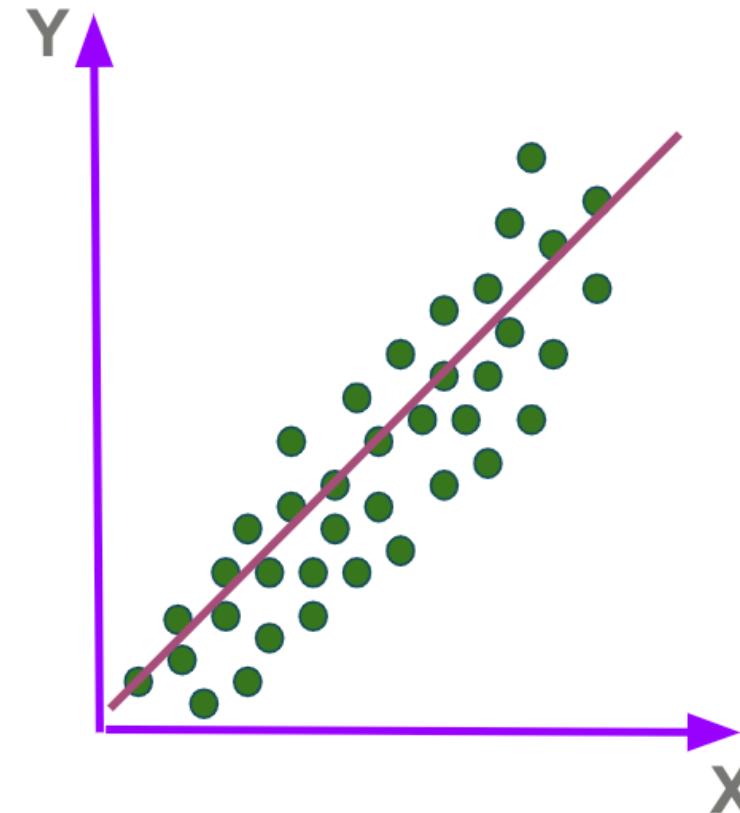
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.2)
print(X_train.shape, X_test.shape)
```

Linear Regression

Correlation



Linear Regression



Linear Regression

$\theta \Rightarrow -5$

$1 \Rightarrow -4.5$

$2 \Rightarrow -4$

$3 \Rightarrow -3.5$

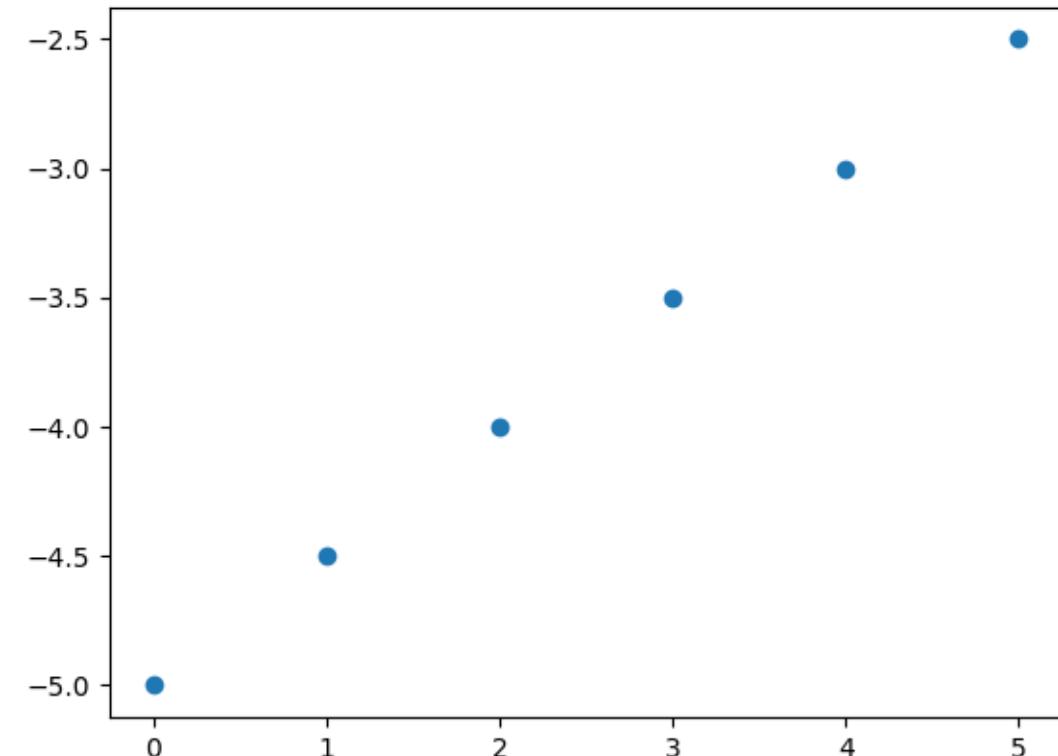
$4 \Rightarrow -3$

$5 \Rightarrow -2.5$

$8 \Rightarrow ?$

Linear Regression

```
import numpy as np  
  
import matplotlib.pyplot as plt  
  
#  $Y = 0.5*X - 5$   
X = np.array([0, 1, 2, 3, 4, 5])  
Y = np.array([-5, -4.5, -4, -3.5, -3, -2.5])  
  
# Plot X and Y  
plt.plot(X, Y, 'o')  
  
# Show the plot  
plt.show()
```



Linear Regression

```
import numpy as np
from sklearn.linear_model import LinearRegression

#Y=0.5*X-5
X=np.array([0,1,2,3,4,5])
Y=np.array([-5,-4.5,-4,-3.5,-3,-2.5])

model = LinearRegression()
model.fit(X,Y)
```

```
import numpy as np
from sklearn.linear_model import LinearRegression

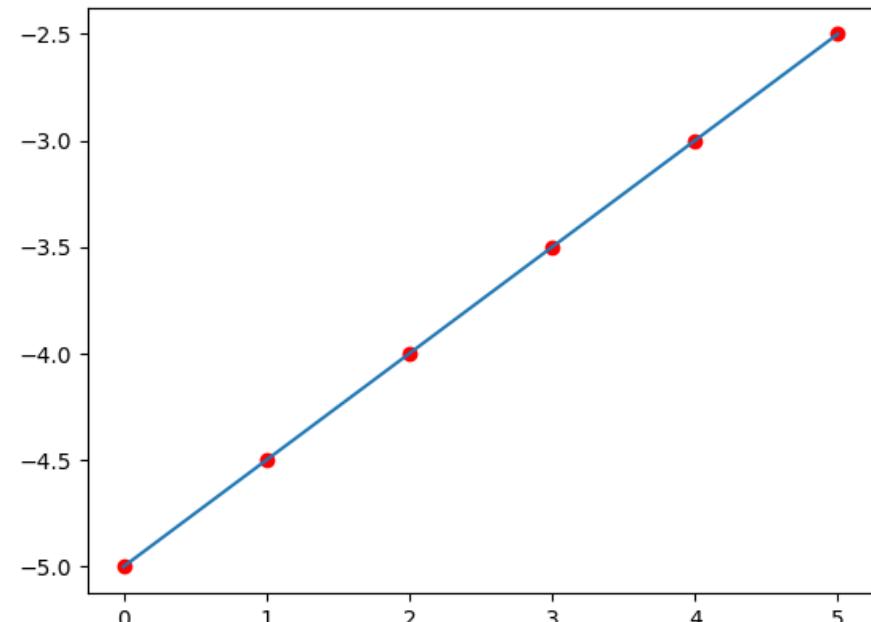
#Y=0.5*X-5
X=np.array([0,1,2,3,4,5]).reshape(-1,1)
Y=np.array([-5,-4.5,-4,-3.5,-3,-2.5])

model = LinearRegression()
model.fit(X,Y)
```

```
ValueError: Expected 2D array, got 1D array instead:
array=[0 1 2 3 4 5].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.resha
pe(1, -1) if it contains a single sample.
```

Linear Regression

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
#Y=0.5*X-5
X=np.array([0,1,2,3,4,5]).reshape(-1,1)
Y=np.array([-5,-4.5,-4,-3.5,-3,-2.5])
plt.plot(X,Y, 'o', color='red')
model = LinearRegression()
model.fit(X,Y)
#y=mx+z
m=model.coef_[0]
z=model.intercept_
print(m,z)
plt.plot(X,model.predict(X))
plt.show()
```



Linear Regression

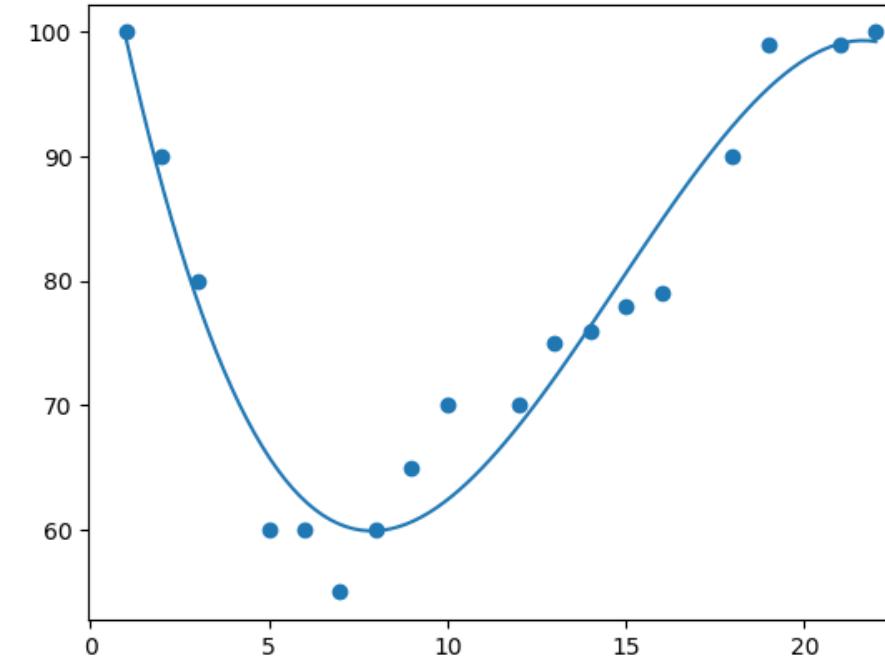
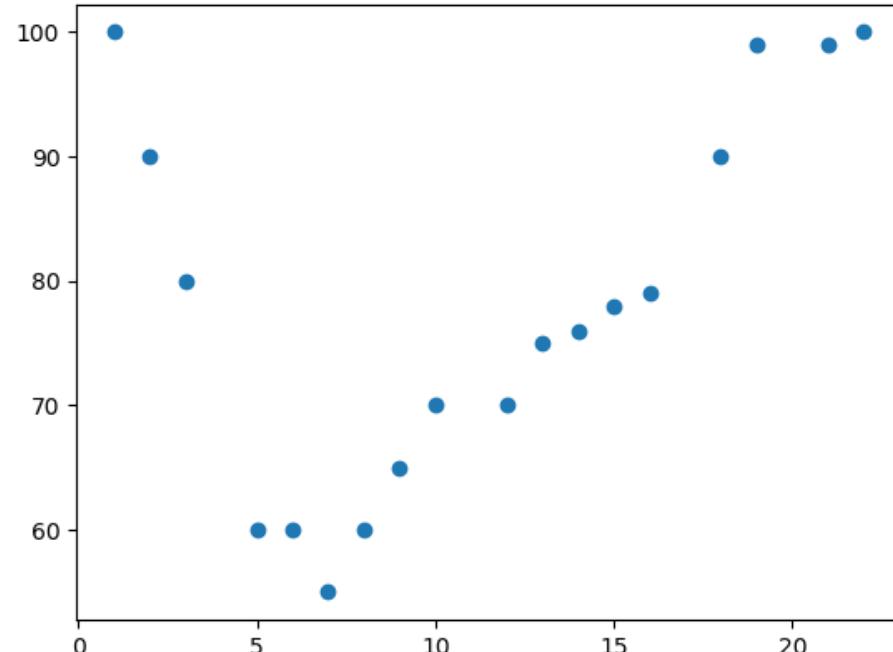
```
import numpy as np
from sklearn.linear_model import LinearRegression

#Y=0.5*X-5
X=np.array([0,1,2,3,4,5]).reshape(-1,1)
Y=np.array([-5,-4.5,-4,-3.5,-3,-2.5])

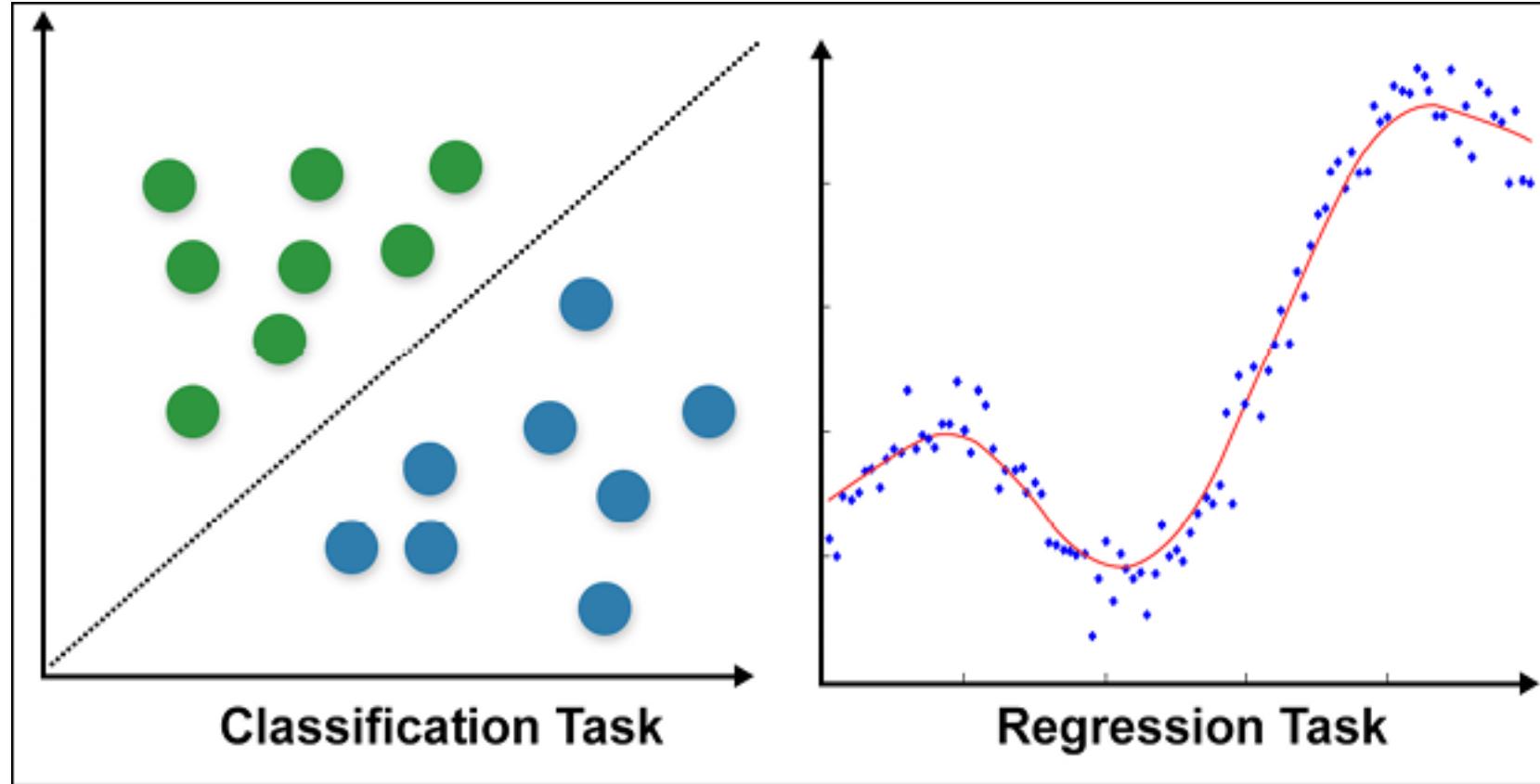
model = LinearRegression()
model.fit(X,Y)

print(model.predict([[8]]))
```

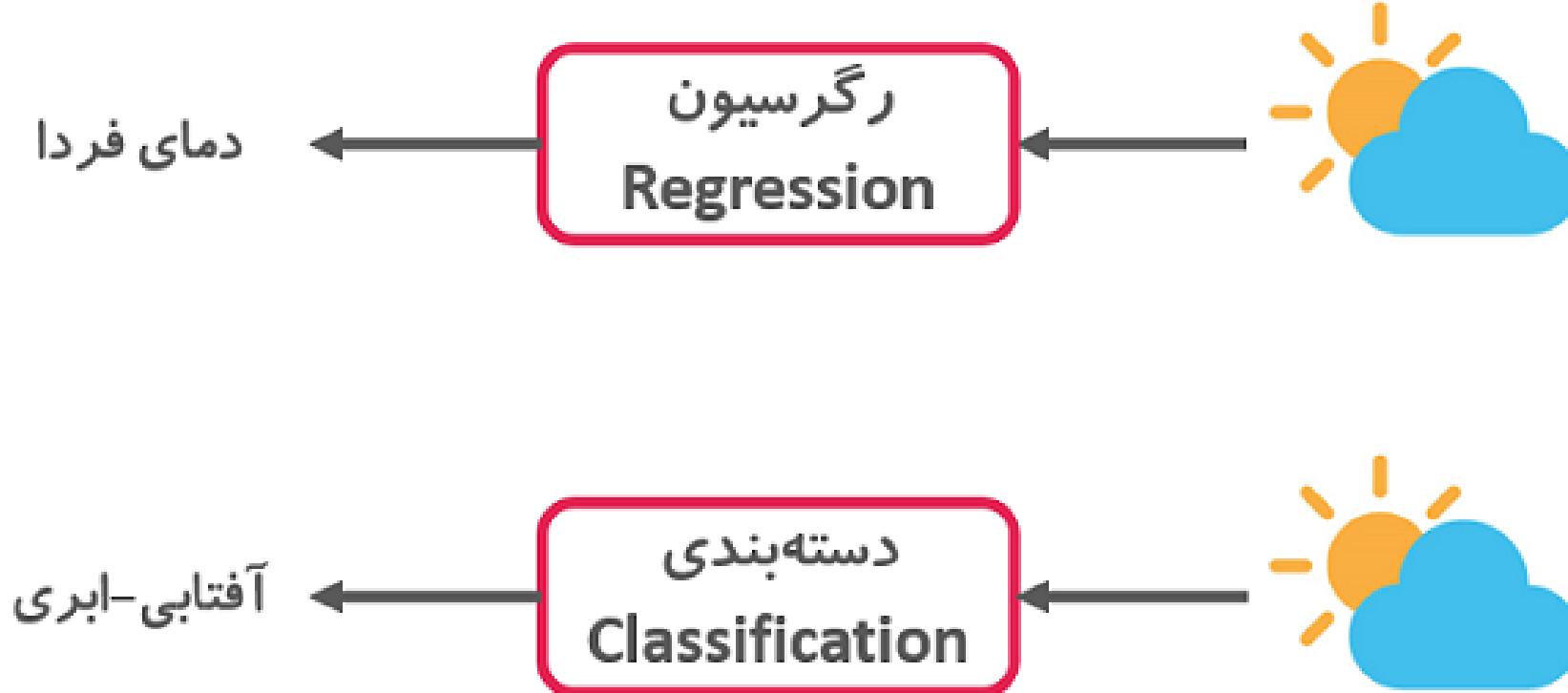
Polynomial Regression



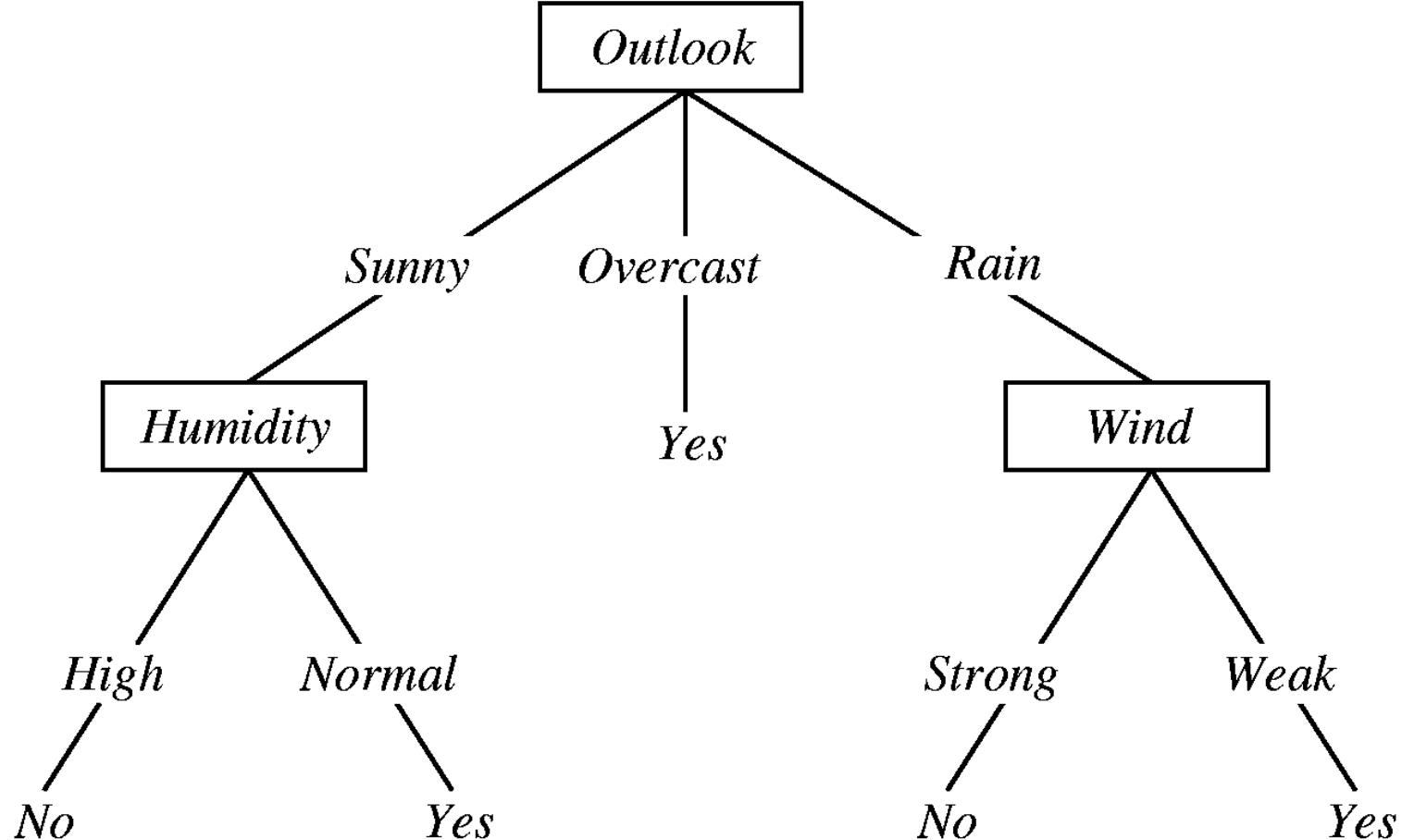
Classification vs Regression



Classification vs Regression

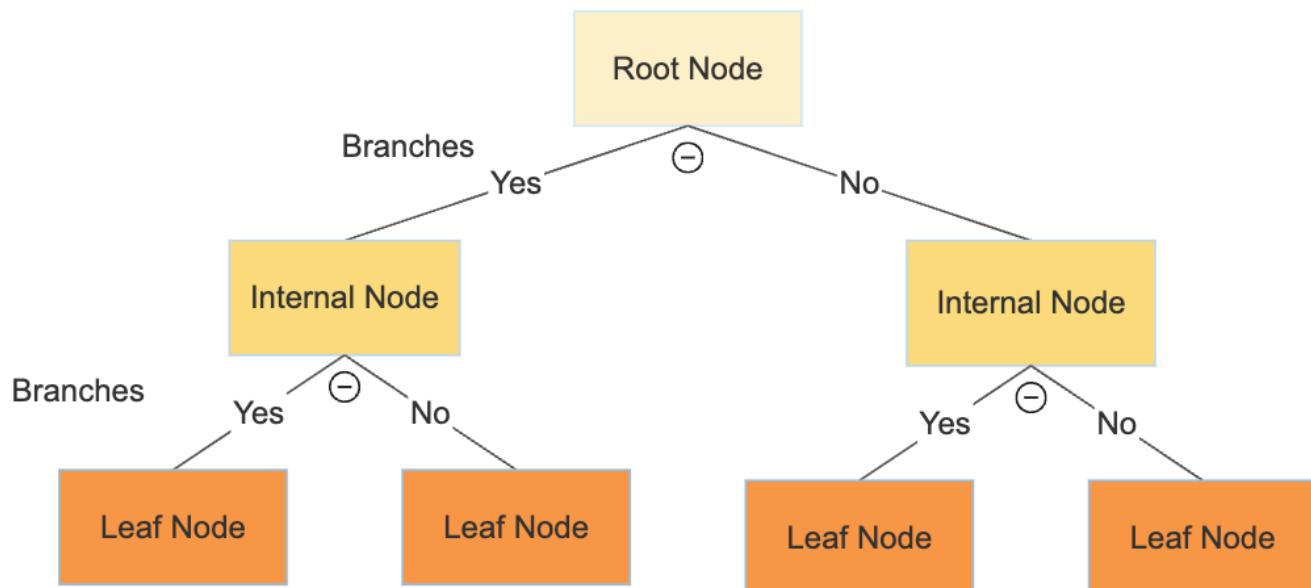


decision tree

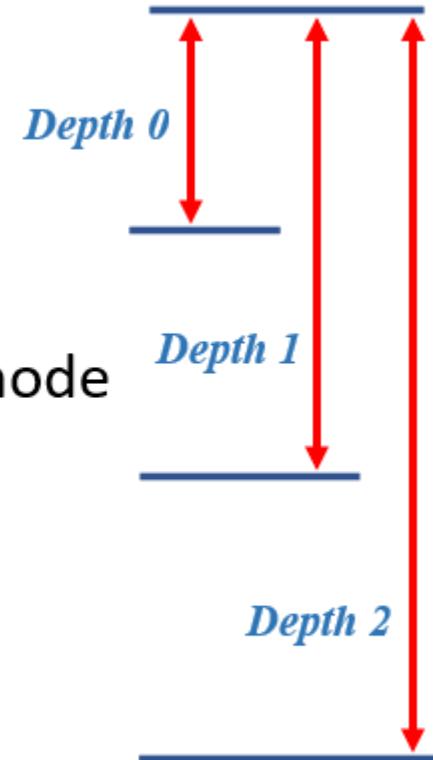
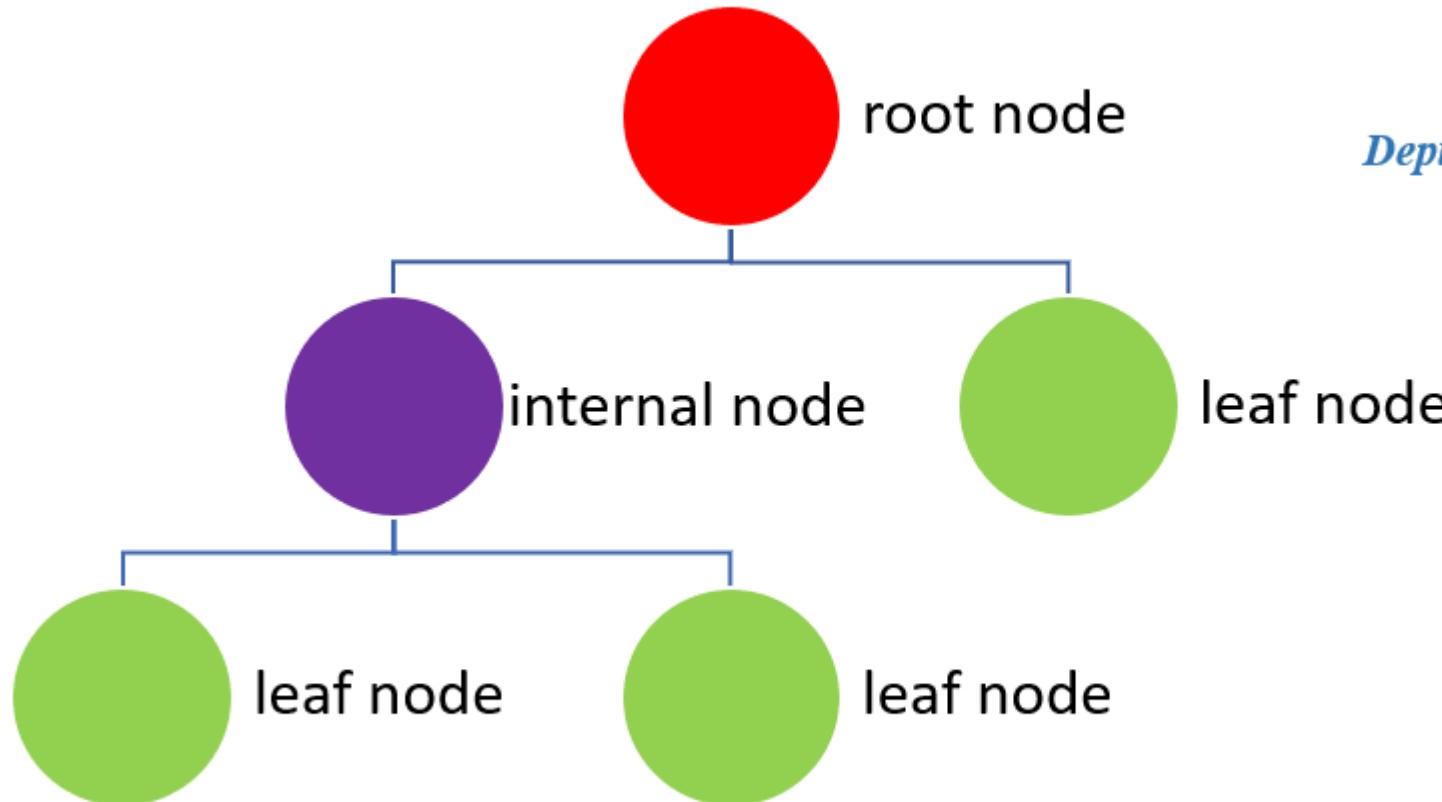


decision tree

Tree



decision tree depth



decision tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Define the model
model = DecisionTreeClassifier(max_depth=10, min_samples_split=4, min_samples_leaf=2)

# Fit the model to the training data
model.fit(X_train, Y_train)

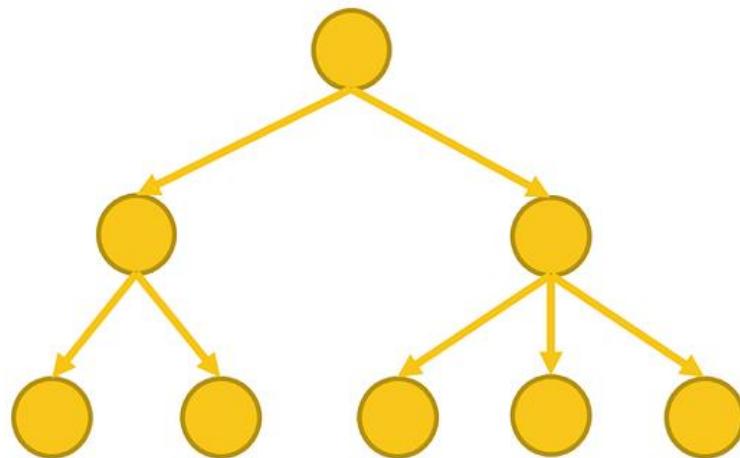
# Make predictions on both the training and test data
Y_train_pred = model.predict(X_train)
Y_test_pred = model.predict(X_test)

# Calculate and print the accuracy for both train and test sets
print('Training accuracy:', accuracy_score(Y_train, Y_train_pred) * 100, '%')
print('Test accuracy:', accuracy_score(Y_test, Y_test_pred) * 100, '%')

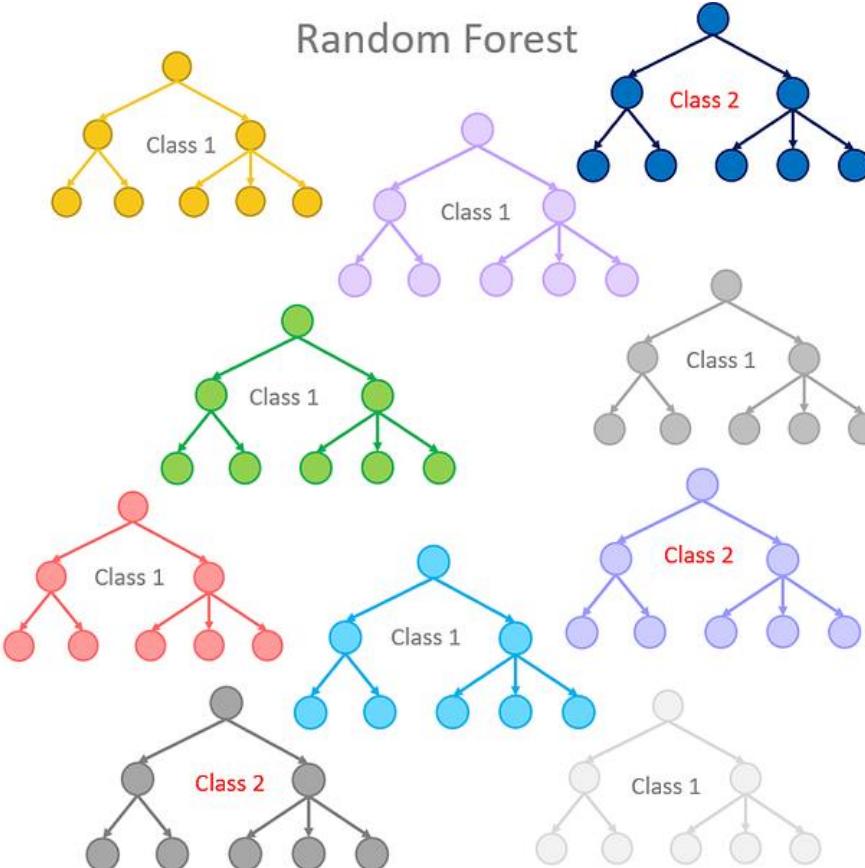
# Make a prediction for a single sample from the training set and reshape it correctly
print('Prediction for sample 10:', model.predict(X_train[10].reshape(1, -1)))
```

Random forest

Single Decision Tree

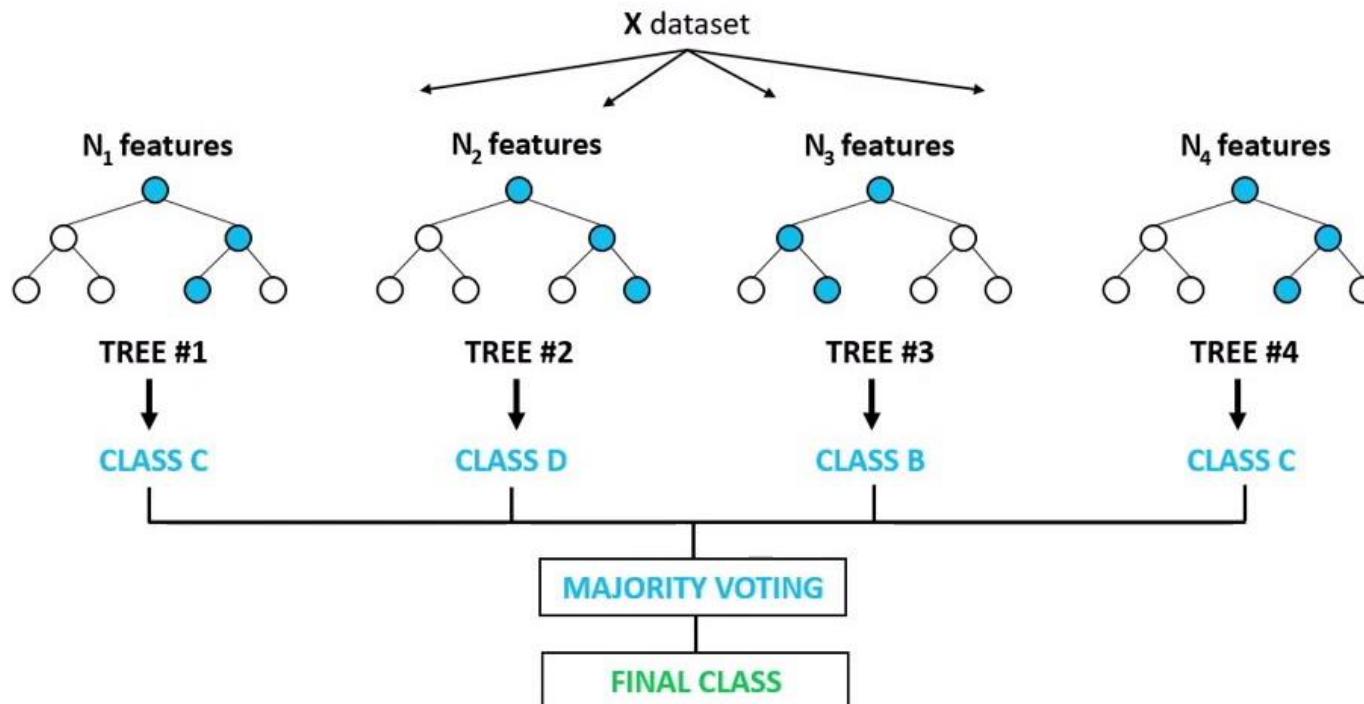


Random Forest



Random forest

Random Forest Classifier



Random forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import numpy as np

# Assuming X_train, Y_train, X_test, and Y_test are defined elsewhere

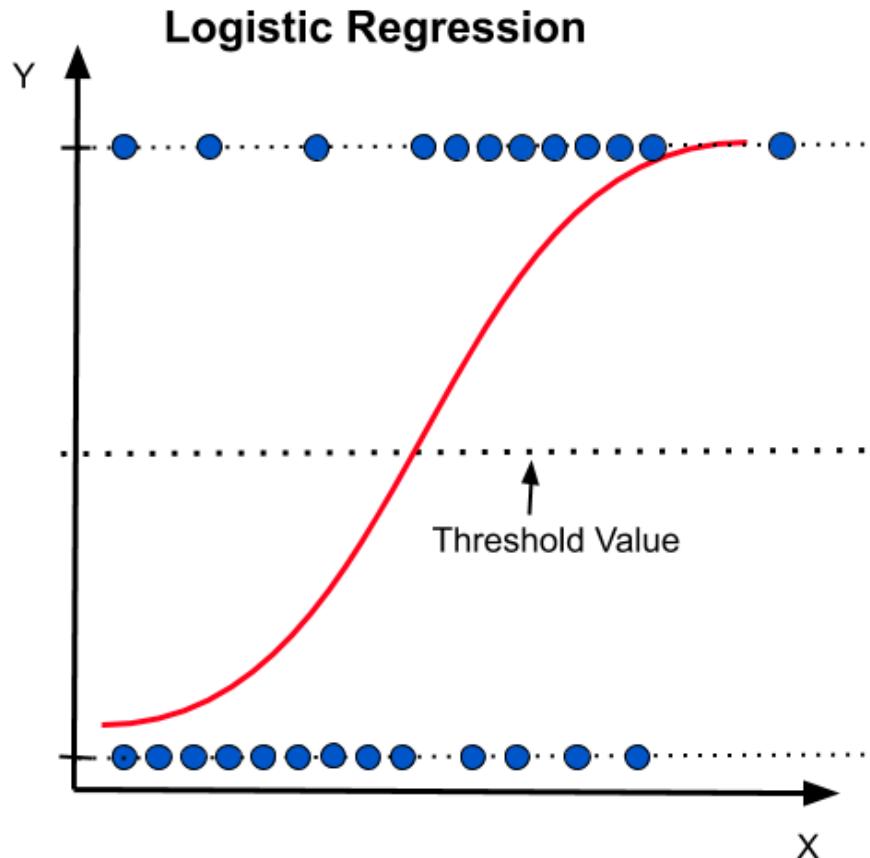
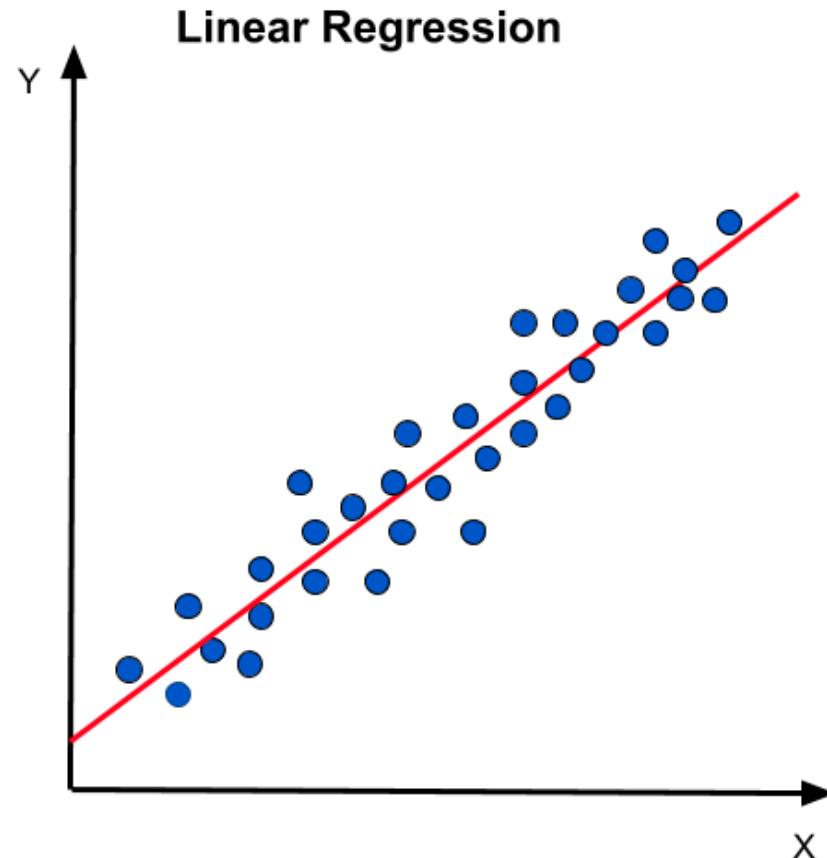
# Create a RandomForestClassifier model with 100 estimators
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, Y_train)

# Predict on the training and test data
Y_train_pred = model.predict(X_train)
Y_test_pred = model.predict(X_test)

# Calculate and print the accuracy for training and test sets
print('Training accuracy:', accuracy_score(Y_train, Y_train_pred) * 100, '%')
print('Test accuracy:', accuracy_score(Y_test, Y_test_pred) * 100, '%')

# Predict on a single instance from X_train (e.g., the 10th instance)
print('Prediction for 10th instance:', model.predict([X_train[10]]))
```

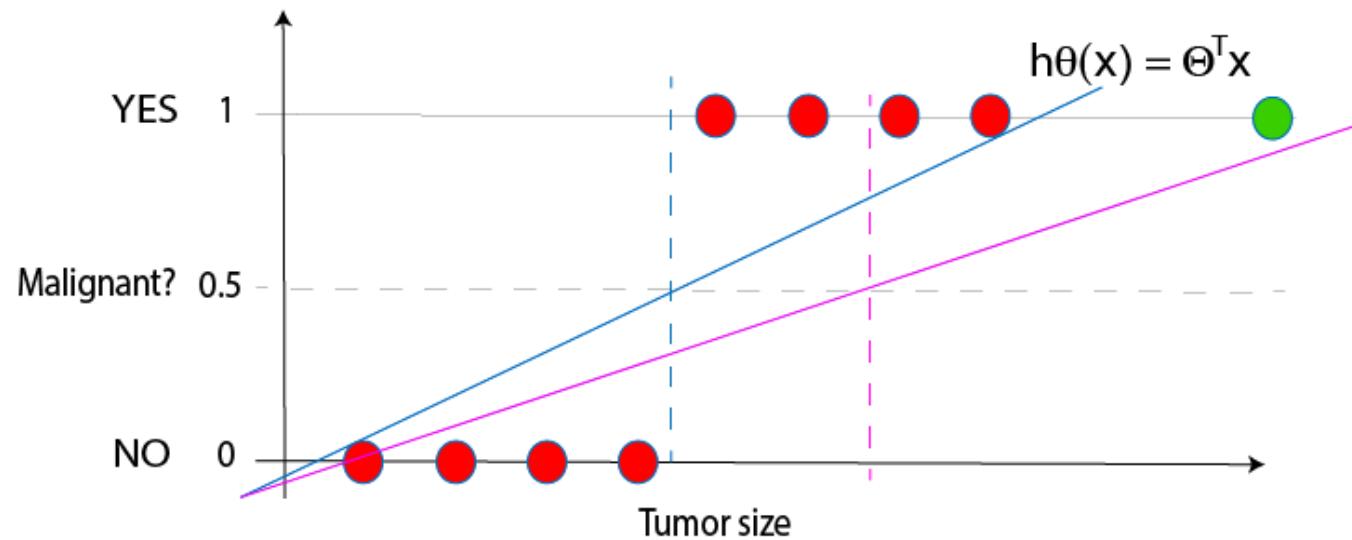
Logistic Regression(binary classification)



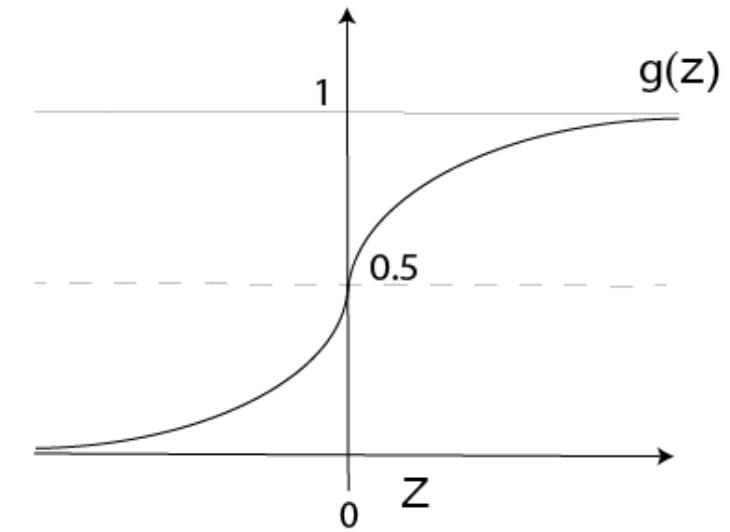
Logistic Regression(binary classification)

Only binary classification

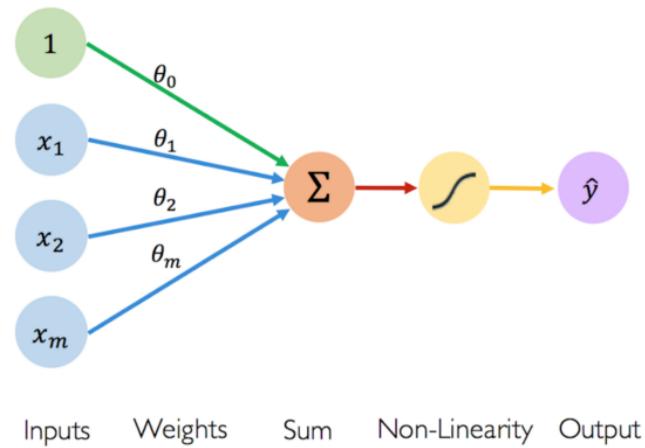
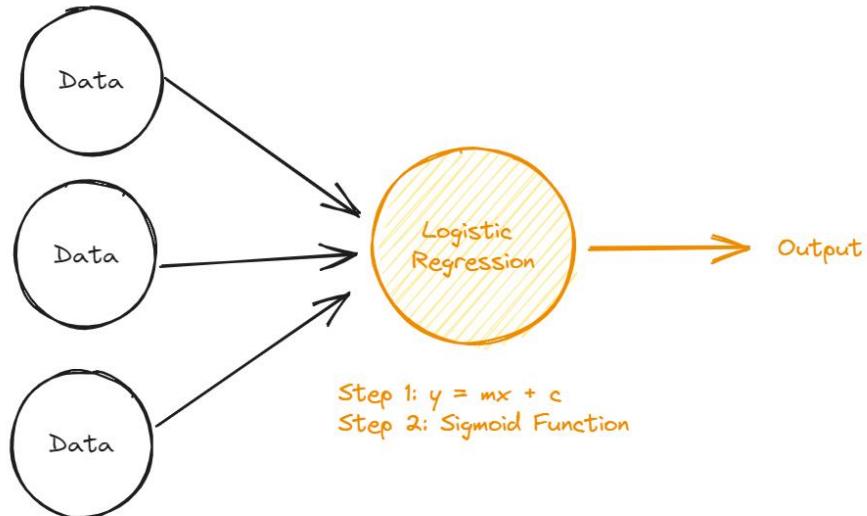
A



B



Logistic Regression(binary classification)

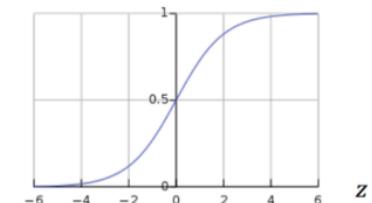


Activation Functions

$$\hat{y} = g(\theta_0 + X^T \theta)$$

- Example: sigmoid function

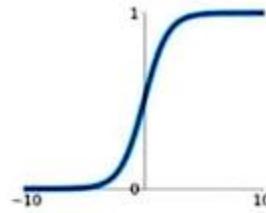
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Activation functions

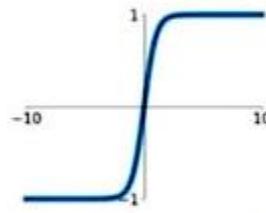
Sigmoid

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



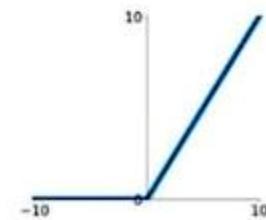
tanh

$$\tanh(x)$$



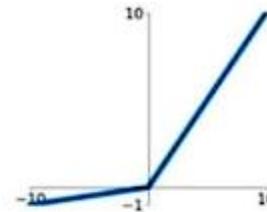
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

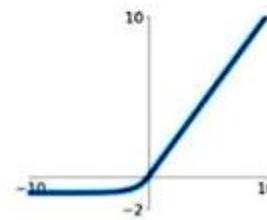


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

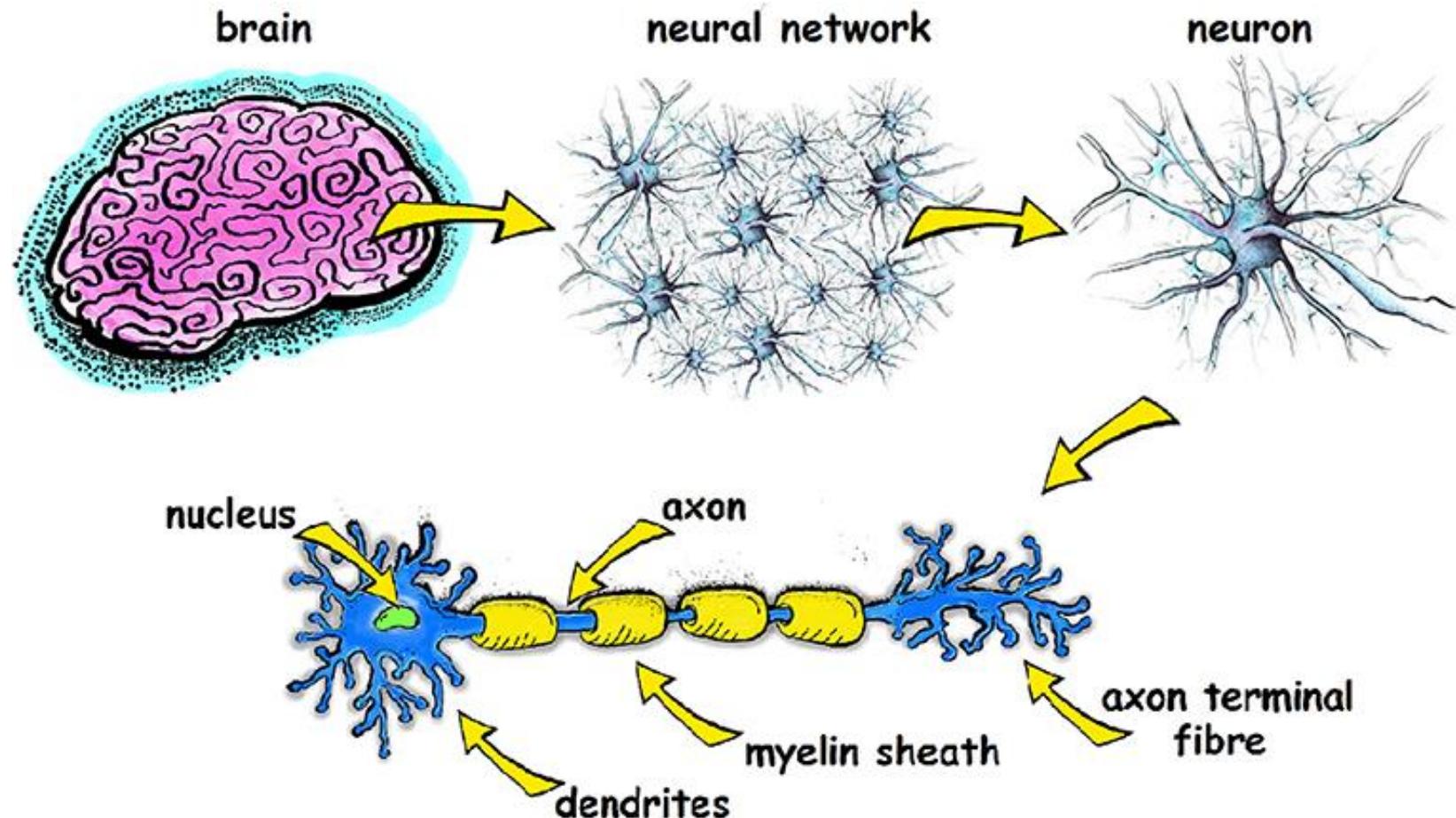
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



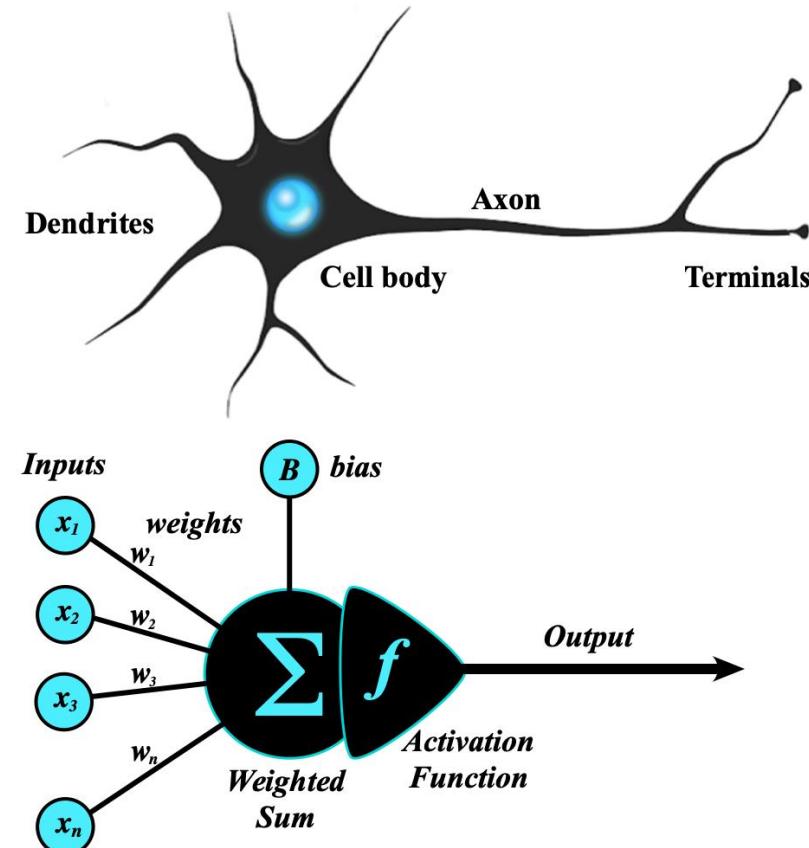
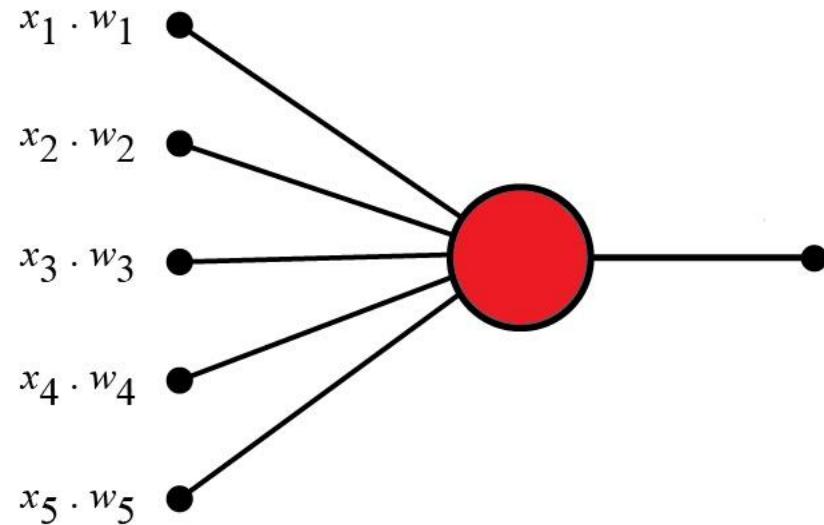
Logistic Regression(binary classification)

```
from sklearn.linear_model import LogisticRegression  
model = LogisticRegression()  
model.fit(X_train, Y_train)
```

neural network



simple neuron



simple neuron

```
import random
class Neuron:
    def __init__(self):
        self.slope = random.random() # Initialize slope randomly
        self.bias = random.random() # Initialize bias randomly

    def forward(self, x):
        # Weighted sum of inputs
        weighted_sum = x * self.slope + self.bias
        # Activation of the weighted sum
        return weighted_sum

    # Function to generate training data
    def generate_data(x, y):
        data = []
        for i in range(len(x)):
            data.append((x[i], y[i])) # Generate data as (x, y)
        return data

    # Train the neuron
    def train_neuron(neuron, data, learning_rate, epochs):
        for _ in range(epochs):
            for x, y in data:
                # Forward pass
                output = neuron.forward(x)
                # Update weights based on error
                neuron.slope += learning_rate * (y - output) * x
                neuron.bias += learning_rate * (y - output)

    # Given values of x and y
    x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    y = [-2 * xi - 5 for xi in x]

    # Create neuron
    neuron = Neuron()

    # Generate training data
    data = generate_data(x, y)

    # Train the neuron
    learning_rate = 0.01
    epochs = 10000
    train_neuron(neuron, data, learning_rate, epochs)

    # Print the learned slope and bias
    print("Learned Slope (m):", neuron.slope)
    print("Learned Bias (b):", neuron.bias)
```

```
import random

class Neuron:
    def __init__(self, weight, bias):
        self.weight = weight
        self.bias = bias

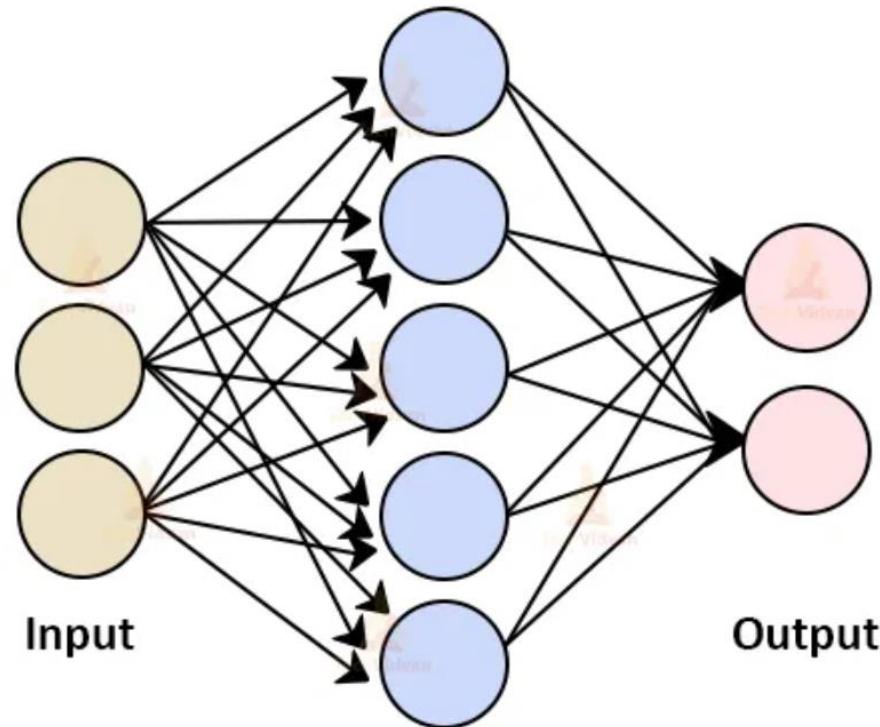
    def forward(self, input_val):
        # Weighted sum of input
        return input_val * self.weight + self.bias

    # Example usage:
    if __name__ == "__main__":
        # Create neuron with weight and bias
        weight = random.random()
        bias = 0
        neuron = Neuron(weight, bias)

        # Input value
        input_val = 0.5

        # Perform forward pass
        output = neuron.forward(input_val)
        print("Output:", output)
```

neural network



neural network

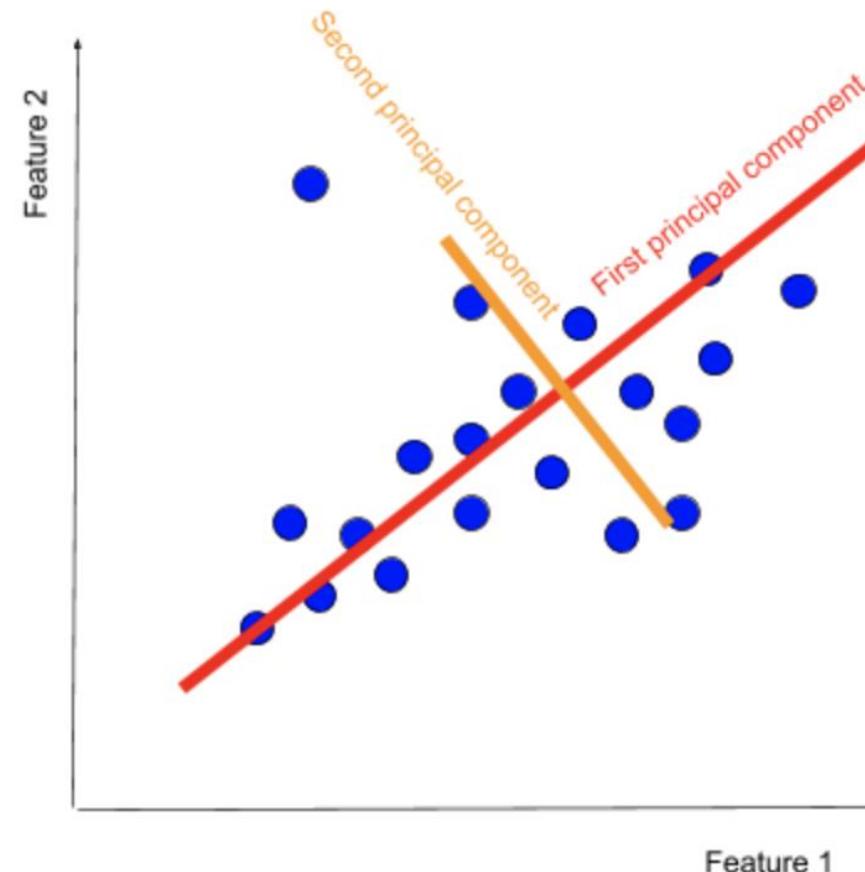
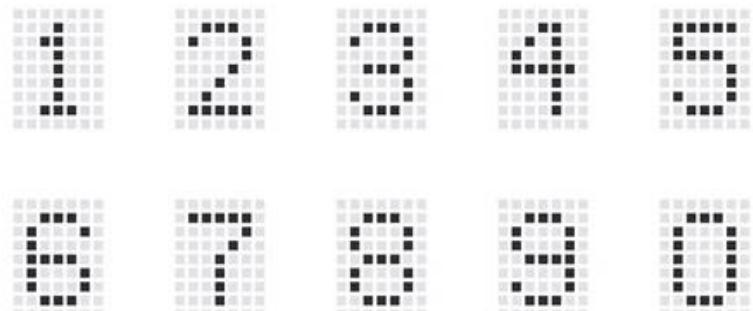
```
from sklearn.neural_network import MLPClassifier

# Initialize the MLPClassifier with a single hidden Layer of 64 neurons
model = MLPClassifier(hidden_layer_sizes=64, max_iter=250)

# Fit the model to the training data
model.fit(X_train, Y_train)
```

PCA(Principal Component Analysis)

| | A | B | C | D | E |
|----|------|----------|---------|-----|---------|
| 1 | area | bedrooms | balcony | age | price |
| 2 | 1200 | 2 | 0 | 2 | 500000 |
| 3 | 2300 | 3 | 2 | 5 | 620000 |
| 4 | 2500 | 4 | 2 | 1 | 122500 |
| 5 | 3650 | 5 | 3 | 3 | 6000000 |
| 6 | 1800 | 3 | 1 | 5 | 2122000 |
| 7 | 3000 | 3 | 1 | 4 | 120000 |
| 8 | 1222 | 1 | 0 | 2 | 450000 |
| 9 | 4600 | 5 | 3 | 1 | 6500000 |
| 10 | 2050 | 2 | 2 | 2 | 1530000 |
| 11 | 1450 | 2 | 2 | 3 | 1563330 |



PCA(Principal Component Analysis)

```
from sklearn.decomposition import PCA  
  
pca = PCA(n_components=6)  
X_train = pca.fit_transform(X_train)  
X_test = pca.transform(X_test)
```

neural network (Polynomial Regression)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X = np.arange(-10, 10, 0.05).reshape(-1, 1)
y = 2*X**4+8*X**3 + X**2 + X + 1
noise = np.random.normal(0, 80, y.shape)
y_noisy = y + noise

X_train, X_test, y_train, y_test = train_test_split(X, y_noisy, test_size=0.2)
mlp = MLPRegressor(hidden_layer_sizes=(128,128,128), activation='relu', max_iter=500)
mlp.fit(X_train, y_train)
y_pred_train = mlp.predict(X_train)
y_pred_test = mlp.predict(X_test)
mse = mean_squared_error(y_test, y_pred_test)
print(f'Mean Squared Error: {mse}')

plt.scatter(X_test, y_test, color='blue', label='Real Data')
plt.scatter(X_test, y_pred_test, color='red', label='Predicted Data')
plt.title('Real vs Predicted Data')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()

plt.scatter(X_train, y_train, color='blue', label='Real Data')
plt.scatter(X_train, y_pred_train, color='red', label='Predicted Data')
plt.title('Real vs Predicted Data')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Open cv(pip install opencv-python)

```
python
import cv2

# Open a connection to the webcam (usually device 0)
cap = cv2.VideoCapture(0)

# Check if the webcam is opened correctly
if not cap.isOpened():
    print("Error: Could not open video stream.")
    exit()

while True:
    # Capture frame-by-frame from the webcam
    ret, frame = cap.read()

    # If frame is captured successfully
    if ret:
        # Display the resulting frame in a window
        cv2.imshow('Webcam Feed', frame)

    # Exit Loop when 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

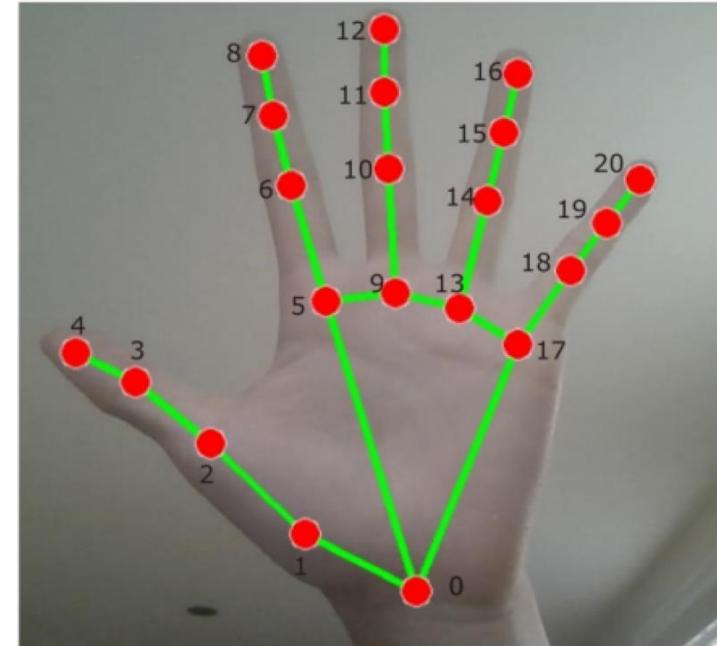
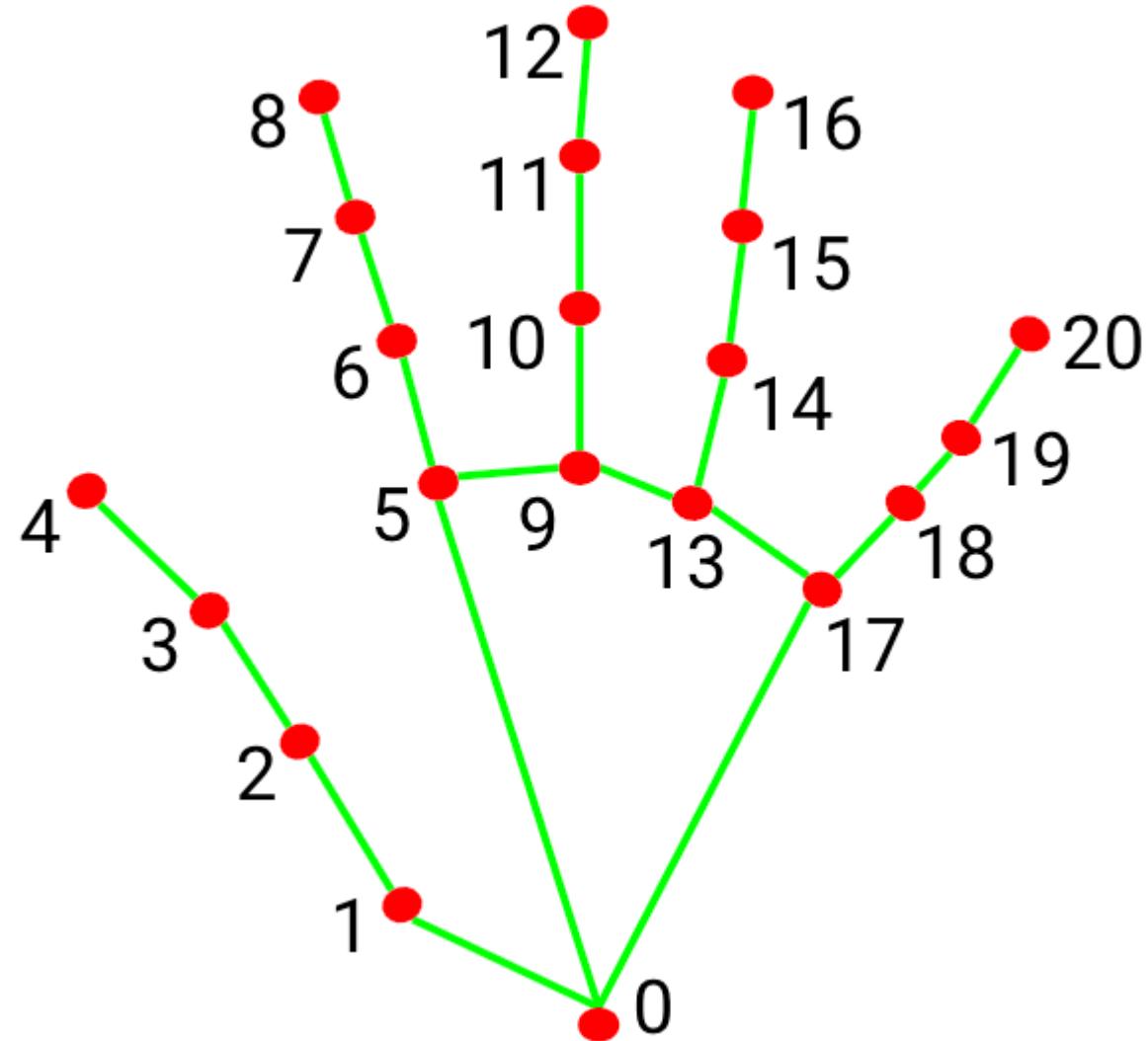
# Release the webcam and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

Copy code

```
# Replace with your IP camera URL
# This might be an RTSP or HTTP URL depending on the camera's streaming protocol
# Example for RTSP: rtsp://username:password@camera_ip:port/stream_path
# Example for HTTP: http://camera_ip:port/video
ip_camera_url = "http://username:password@camera_ip:port/video"

# Open the IP camera video stream
cap = cv2.VideoCapture(ip_camera_url)
```

Media pipe(hand landmarks)



Media pipe(hand landmarks)

```
import cv2
from cvzone.HandTrackingModule import HandDetector

def main():
    # Initialize the webcam to capture video
    cap = cv2.VideoCapture(0)

    # Initialize the HandDetector class with the given parameters
    detector = HandDetector(staticMode=False, maxHands=2, modelComplexity=1, detectionCon=0.5, minTrackCon=0.5)

    # Continuously get frames from the webcam
    while True:
        # Capture each frame from the webcam
        success, img = cap.read()

        # If frame capture was not successful, break the loop
        if not success:
            print("Failed to capture image from camera.")
            break

        # Find hands in the current frame
        hands, img = detector.findHands(img, draw=True, flipType=True)

        # Check if any hands are detected
        if hands:
            # Iterate over detected hands
            for i, hand in enumerate(hands):
                # Count the number of fingers up for the detected hand
                fingers = detector.fingersUp(hand)
                print(f'Hand {i + 1} Fingers Up: {fingers.count(1)}') # Print the count of fingers that are up

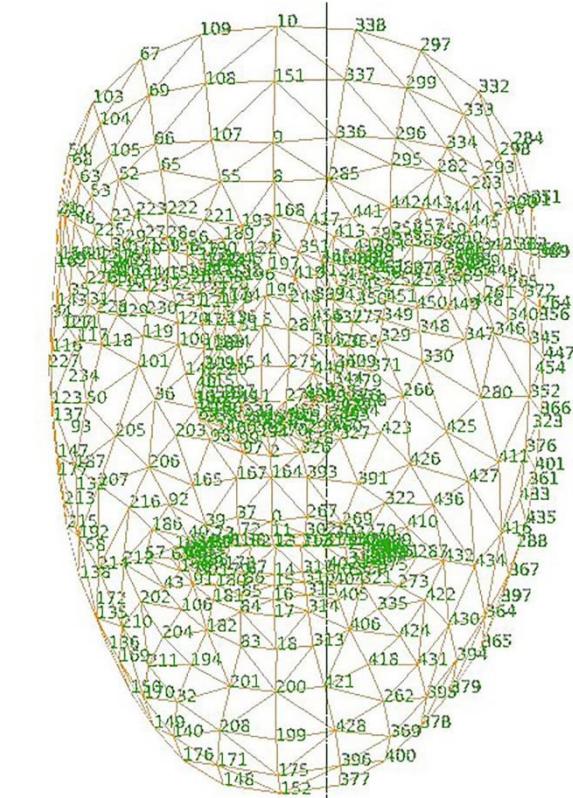
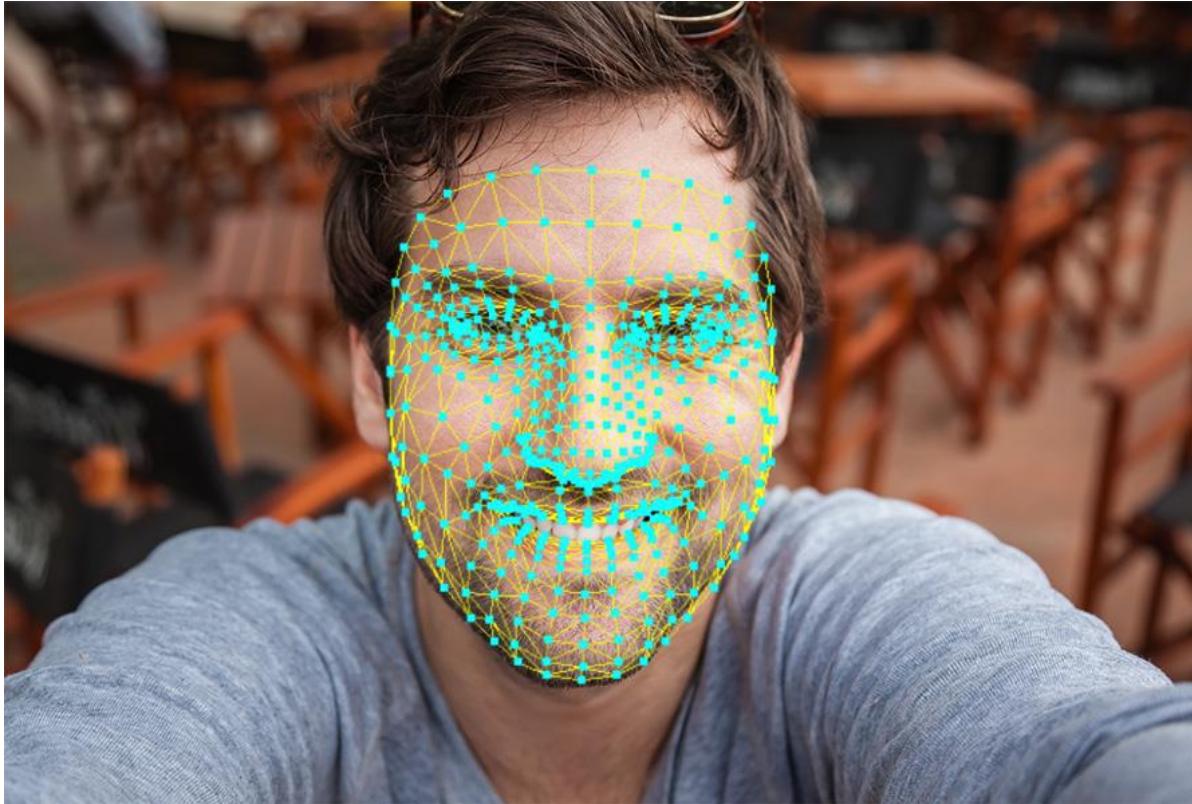
            # Display the image in a window
            cv2.imshow("Hand Tracking", img)

            # Wait for 1 millisecond and check if the user pressed 'q' to exit
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        # Release the video capture and destroy all windows
        cap.release()
        cv2.destroyAllWindows()

    __name__ == "__main__":
        main()
```

Media pipe(face mesh)



Media pipe(face mesh)

```
import cv2
from cvzone.FaceMeshModule import FaceMeshDetector

def main():
    # Initialize the webcam
    cap = cv2.VideoCapture(0)

    # Initialize the FaceMeshDetector with the desired parameters
    detector = FaceMeshDetector(staticMode=False, maxFaces=2, minDetectionCon=0.5, minTrackCon=0.5)

    # Start the loop to continually get frames from the webcam
    while True:
        # Read the current frame from the webcam
        success, img = cap.read()

        # If the frame was not captured successfully, break the loop
        if not success:
            print("Failed to capture image from camera.")
            break

        # Find face mesh in the image
        img, faces = detector.findFaceMesh(img, draw=True)

        # Check if any faces are detected
        if faces:
            # Loop through each detected face
            for face in faces:
                # print(face)
                pass

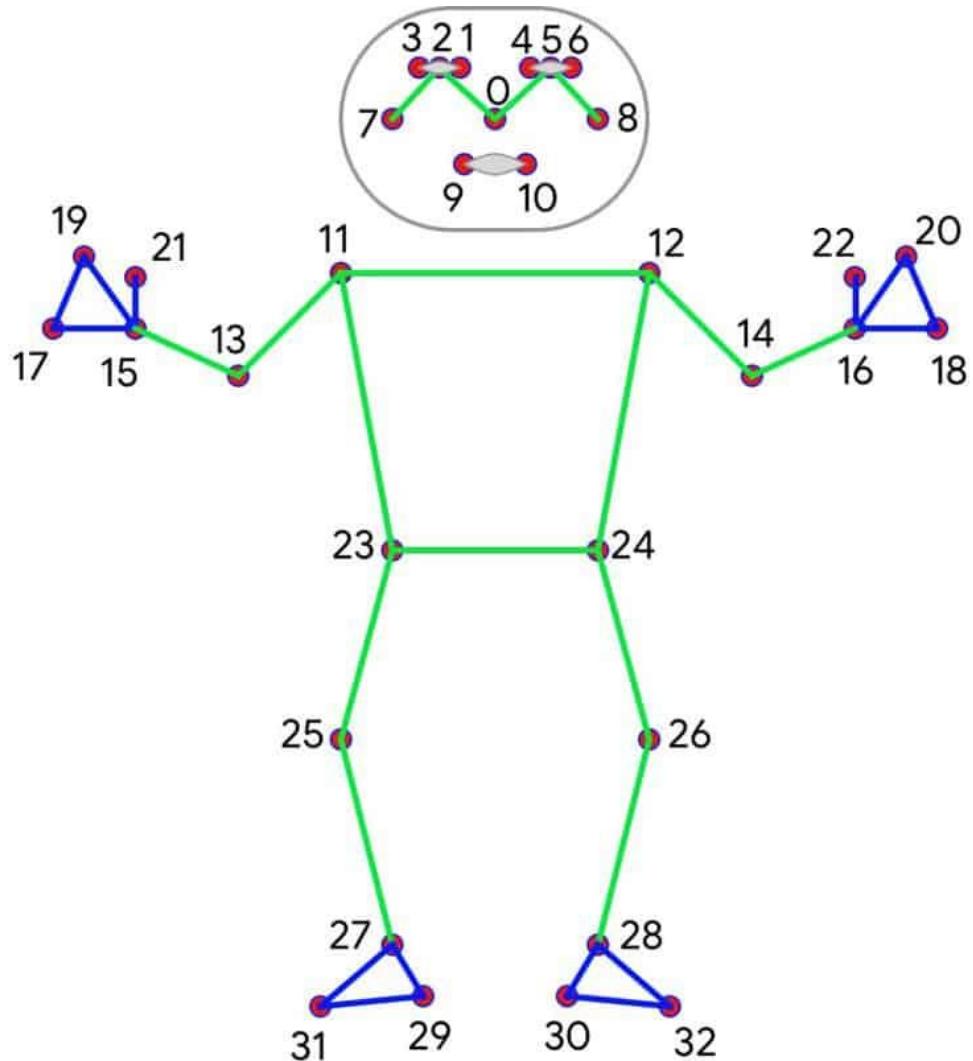
            # Display the image in a window named 'Face Mesh'
            cv2.imshow("Face Mesh", img)

            # Wait for 1 millisecond to check for any user input; if 'q' is pressed, break the loop
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        # Release the webcam and destroy all OpenCV windows
        cap.release()
        cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

Body detection



Body detection

```
import cv2
from cvzone.PoseModule import PoseDetector

def main():
    # Initialize the webcam (change index if needed)
    cap = cv2.VideoCapture(0) # Use 0 for built-in webcam, or another index for external cameras

    # Initialize the PoseDetector from cvzone
    detector = PoseDetector(staticMode=False,
                            modelComplexity=1,
                            smoothLandmarks=True,
                            enableSegmentation=False,
                            smoothSegmentation=True,
                            detectionCon=0.5,
                            trackCon=0.5)

    # Continuously read from the webcam feed
    while True:
        success, img = cap.read() # Read the current frame from the webcam

        if not success:
            print("Error: Could not retrieve frame from webcam.")
            break

        # Find pose landmarks and draw them on the image
        img = detector.findPose(img, draw=True)

        # Get landmark positions (this line is not strictly necessary unless you need the positions)
        lmList, bboxInfo = detector.findPosition(img, draw=True, bboxWithHands=False)

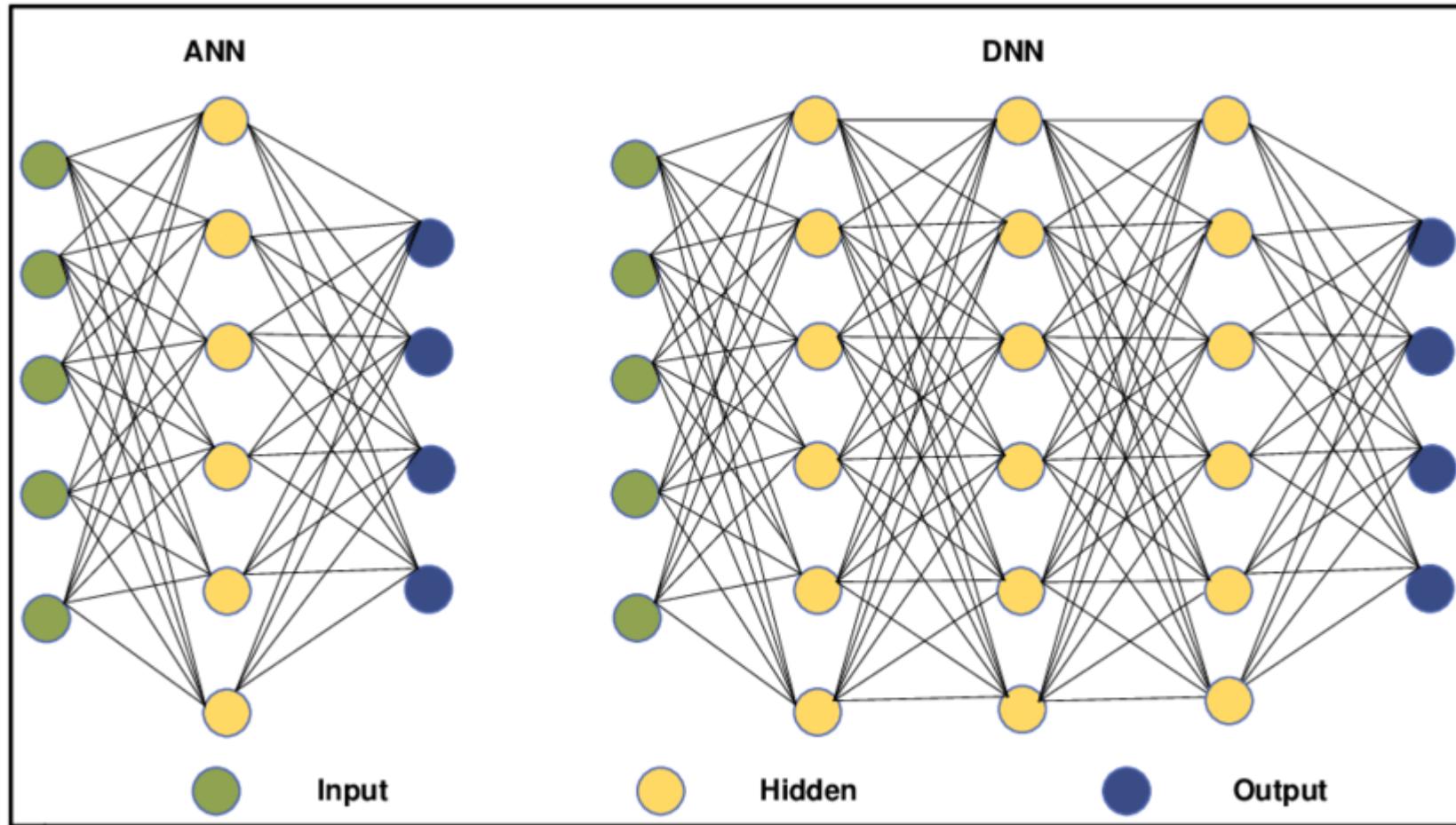
        # Display the processed image with landmarks
        cv2.imshow("Pose Detection", img)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the webcam and close all windows
    cap.release()
    cv2.destroyAllWindows()

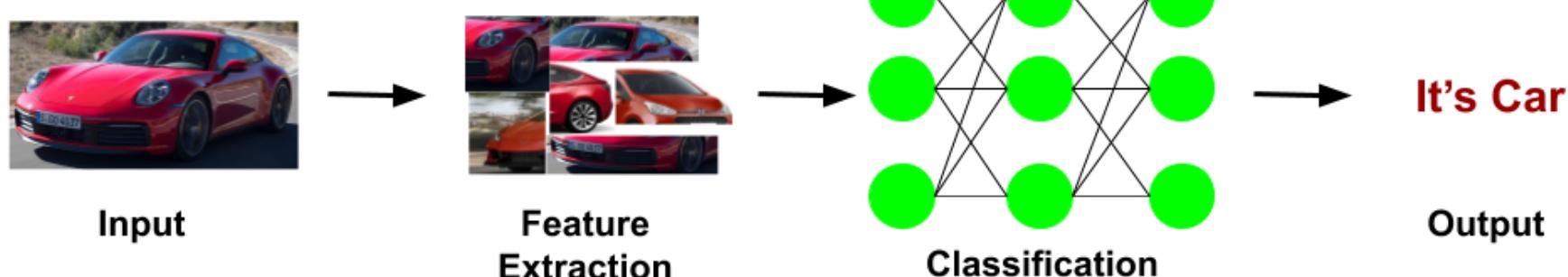
if __name__ == "__main__":
    main()
```

ANN vs DNN

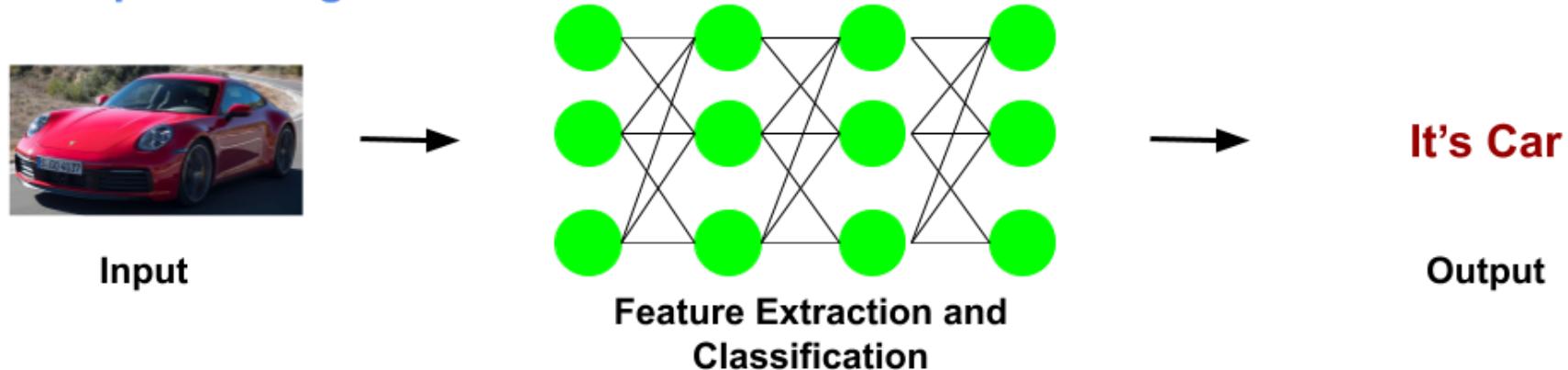


ANN vs DNN

Machine Learning



Deep Learning



Best python libraries for ML



TensorFlow

```
[8] import tensorflow as tf
```

- ▼ 0D,1D,2D and 3D

```
[3] a=tf.constant(5)  
a
```

```
→ <tf.Tensor: shape=(), dtype=int32, numpy=5>
```

```
[5] b=tf.constant([5,2])  
b
```

```
→ <tf.Tensor: shape=(2,), dtype=int32, numpy=array([5, 2], dtype=int32)>
```

```
[6] c=tf.constant([[5,2],[3,4]])  
c
```

```
→ <tf.Tensor: shape=(2, 2), dtype=int32, numpy=  
array([[5, 2],  
       [3, 4]], dtype=int32)>
```

```
[7] d=tf.constant([[5,2],[3,4]],[[7,8],[0,9]])  
d
```

```
→ <tf.Tensor: shape=(2, 2, 2), dtype=int32, numpy=  
array([[[5, 2],  
       [3, 4]],
```

```
      [[7, 8],  
       [0, 9]]], dtype=int32)>
```

TensorFlow

```
[14] t1=tf.constant([5,2])
     t2=tf.constant([6,8])

[15] tf.add(t1,t2)
→ <tf.Tensor: shape=(2,), dtype=int32, numpy=array([11, 10], dtype=int32)>

[16] tf.subtract(t1,t2)
→ <tf.Tensor: shape=(2,), dtype=int32, numpy=array([-1, -6], dtype=int32)>

[17] t3=tf.constant([[5,2],[3,4]])
     t4=tf.constant([[8,9],[10,5]])

[18] tf.multiply(t3,t4)
→ <tf.Tensor: shape=(2, 2), dtype=int32, numpy=
   array([[40, 18],
          [30, 20]], dtype=int32)>

[19] tf.matmul(t3,t4)
→ <tf.Tensor: shape=(2, 2), dtype=int32, numpy=
   array([[60, 55],
          [64, 47]], dtype=int32)>

[21] #broadcasting
     t5=tf.constant([5,2])
     a=tf.constant(5)
     t5*a
→ <tf.Tensor: shape=(2,), dtype=int32, numpy=array([25, 10], dtype=int32)>
```

TensorFlow

▼ reshape

```
[22] t6=tf.constant([6,8,10,12,14,16])  
      t6.shape
```

```
→ TensorShape([6])
```

```
▶ tf.reshape(t6,(2,3))
```

```
→ <tf.Tensor: shape=(2, 3), dtype=int32, numpy=  
array([[ 6,  8, 10],  
       [12, 14, 16]], dtype=int32)>
```

How to make a deep neural network using TF and keras

```
[1] import tensorflow as tf
from tensorflow import keras

[2] x=tf.constant([1,2,3,4,5,6,7,8,9,10])
y=x*5-10
y

→ <tf.Tensor: shape=(10,), dtype=int32, numpy=array([-5,  0,  5, 10, 15, 20, 25, 30, 35, 40], dtype=int32)>

▶ model=keras.Sequential([
    keras.layers.Dense(units=1,activation=None,input_shape=[1])
])

[7] model.compile(optimizer='sgd',loss='mean_squared_error')

[5] model.summary()

→ Model: "sequential"
    ┌─────────┐
    │ Layer (type)      ──┐
    │ Output Shape     ─┤
    └─────────┘
    dense (Dense)    ─┤ (None, 1)
                      ─┤
                      └─ 2

    Total params: 2 (8.00 B)
    Trainable params: 2 (8.00 B)
    Non-trainable params: 0 (0.00 B)

▶ model.fit(x,y,batch_size=1,epochs=100)

→ Epoch 1/100
10/10 ━━━━━━━━ 1s 3ms/step - loss: 268.4004
Epoch 2/100
10/10 ━━━━━━ 0s 2ms/step - loss: 51.6184
Epoch 3/100
10/10 ━━━━ 0s 2ms/step - loss: 34.5277
Epoch 4/100
10/10 ━━ 0s 2ms/step - loss: 26.5179
Epoch 5/100
10/10 ━ 0s 2ms/step - loss: 14.2520
```

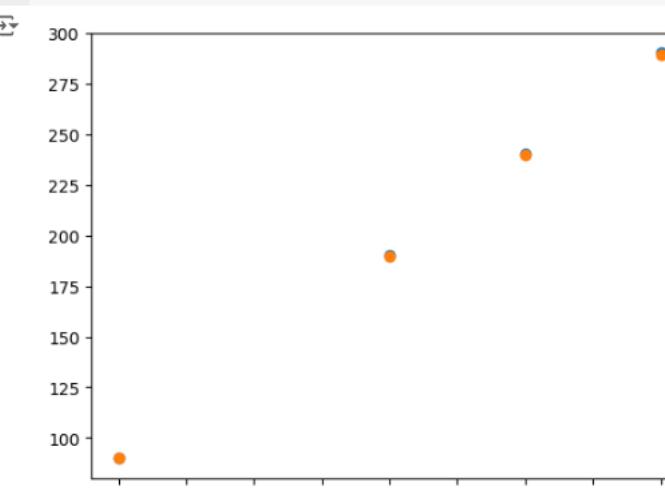
How to make a deep neural network using TF and keras

```
[26] x_test=tf.constant([20,40,50,60],dtype=float)
      x_test
      ↵ <tf.Tensor: shape=(4,), dtype=float32, numpy=array([20., 40., 50., 60.], dtype=float32)>

[28] y_test=x_test**5-10
      y_test
      ↵ <tf.Tensor: shape=(4,), dtype=float32, numpy=array([-90., -190., -240., -290.], dtype=float32)>

[29] y_pred=model.predict(x_test)
      y_pred
      ↵ 1/1 ━━━━━━ 0s 17ms/step
      array([[ 89.84898],
             [189.56973],
             [239.4301 ],
             [289.29047]], dtype=float32)

❶ import matplotlib.pyplot as plt
❷ plt.scatter(x_test,y_test)
❸ plt.scatter(x_test,y_pred)
❹ plt.show()
```



| x | y | pred |
|----|-----|-----------|
| 20 | 90 | 89.84898 |
| 40 | 190 | 189.56973 |
| 50 | 240 | 239.4301 |
| 60 | 290 | 289.29047 |

Comparison of deep neural network and neural network

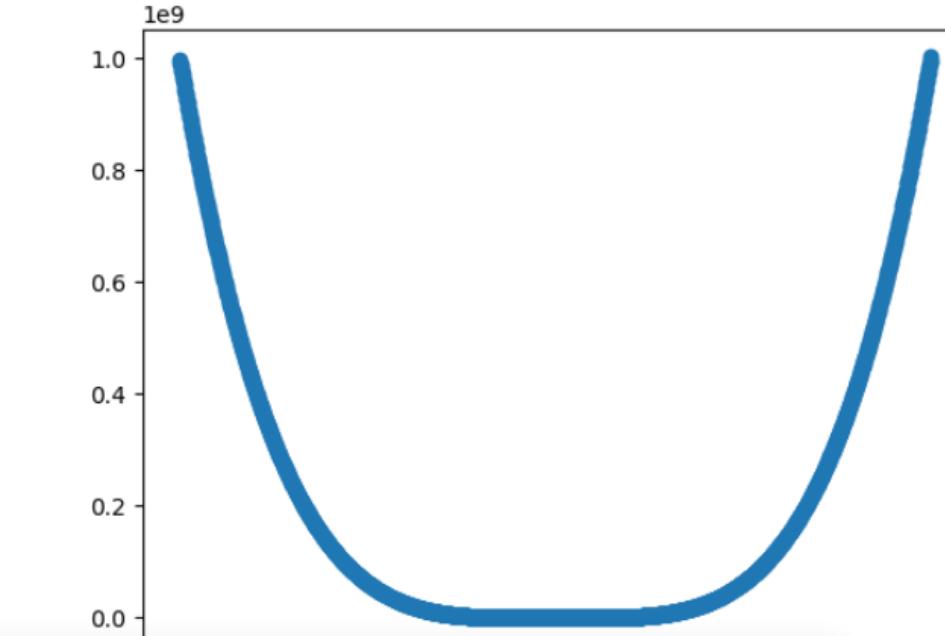
```
[1] import numpy as np
    import matplotlib.pyplot as plt

[2] x=np.arange(-100,100,0.1)
    x

→ array([-100. , -99.9, -99.8, ..., 99.7, 99.8, 99.9])

▶ y=10*x**4+5*x**3+2*x**2+x+5
    plt.scatter(x,y)

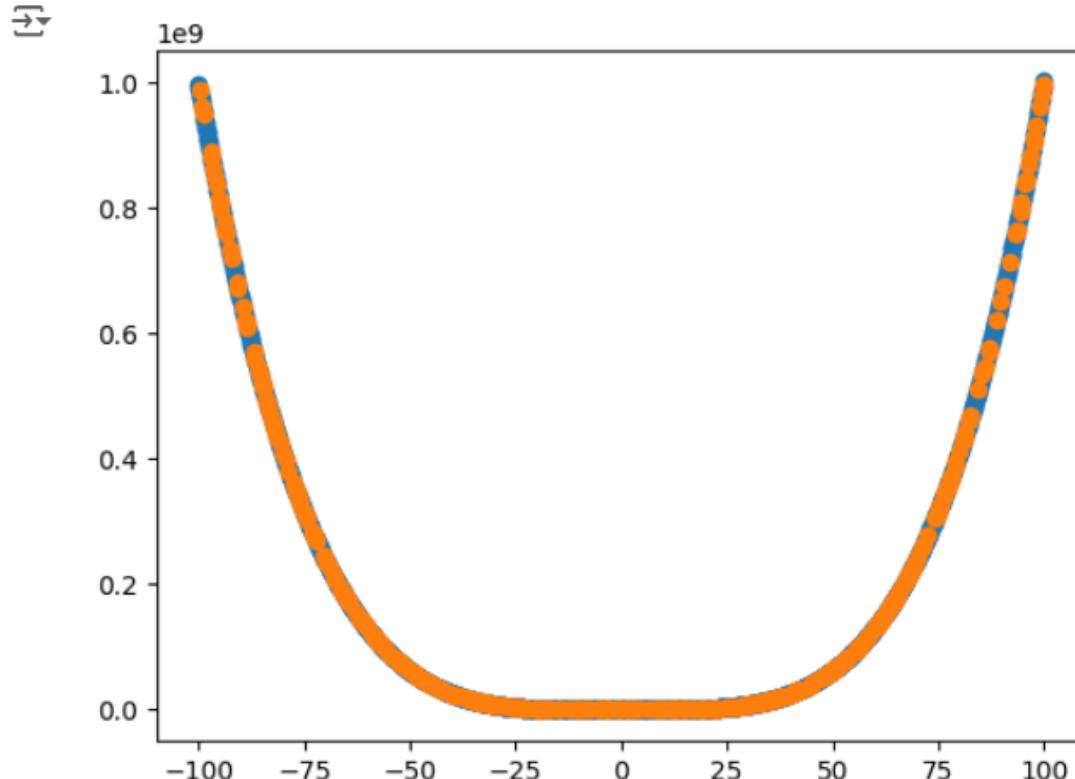
→ <matplotlib.collections.PathCollection at 0x7a1a0526b700>
```



Comparison of deep neural network and neural network

```
[19] from sklearn.model_selection import train_test_split  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

```
▶ plt.scatter(x_train,y_train)  
plt.scatter(x_test,y_test)  
plt.show()
```



Comparison of deep neural network and neural network

```
[55] import tensorflow as tf
     from tensorflow import keras
     model=keras.Sequential()
     model.add(keras.layers.Dense(1,input_shape=(1,)))
     model.add(keras.layers.Dense(128,activation='relu'))
     model.add(keras.layers.Dense(128,activation='relu'))
     model.add(keras.layers.Dense(256,activation='relu'))
     model.add(keras.layers.Dense(128,activation='relu'))

     model.add(keras.layers.Dense(1))

[56] model.compile(optimizer='adam',loss='mean_squared_error',metrics=['accuracy'])

▶ model.summary()

→ Model: "sequential_9"

```

| Layer (type) | Output Shape | Param # |
|------------------|--------------|---------|
| dense_50 (Dense) | (None, 1) | 2 |
| dense_51 (Dense) | (None, 128) | 256 |
| dense_52 (Dense) | (None, 128) | 16,512 |
| dense_53 (Dense) | (None, 256) | 33,024 |
| dense_54 (Dense) | (None, 128) | 32,896 |
| dense_55 (Dense) | (None, 1) | 129 |

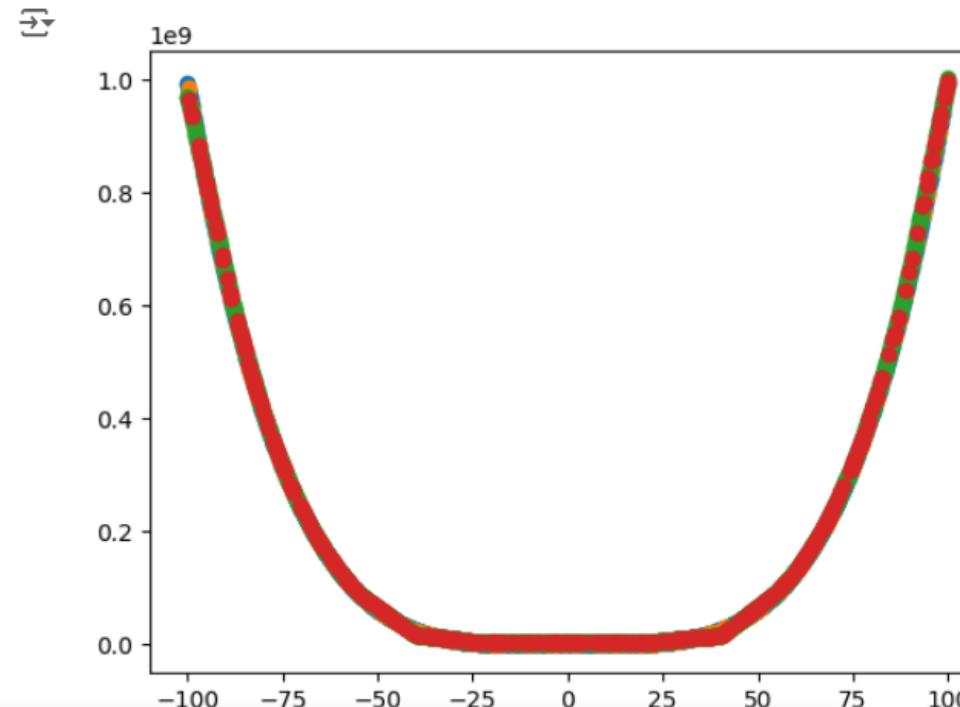
```
Total params: 82,819 (323.51 KB)
Trainable params: 82,819 (323.51 KB)
Non-trainable params: 0 (0.00 B)
```

Comparison of deep neural network and neural network

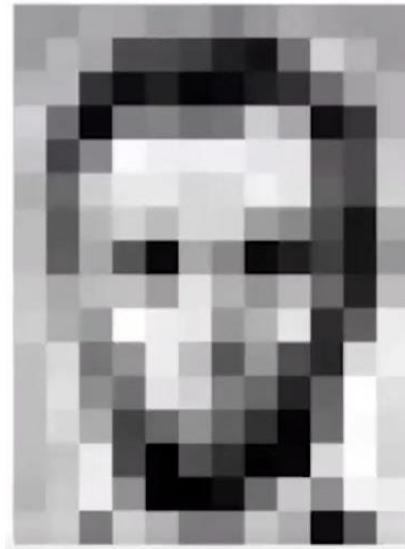
```
[58] model.fit(x_train,y_train,batch_size=5,epochs=100)
```

```
▶ y_pred_train=model.predict(x_train)  
y_pred_test=model.predict(x_test)
```

```
▶ plt.scatter(x_train,y_train)  
plt.scatter(x_test,y_test)  
plt.scatter(x_train,y_pred_train)  
plt.scatter(x_test,y_pred_test)  
plt.show()
```

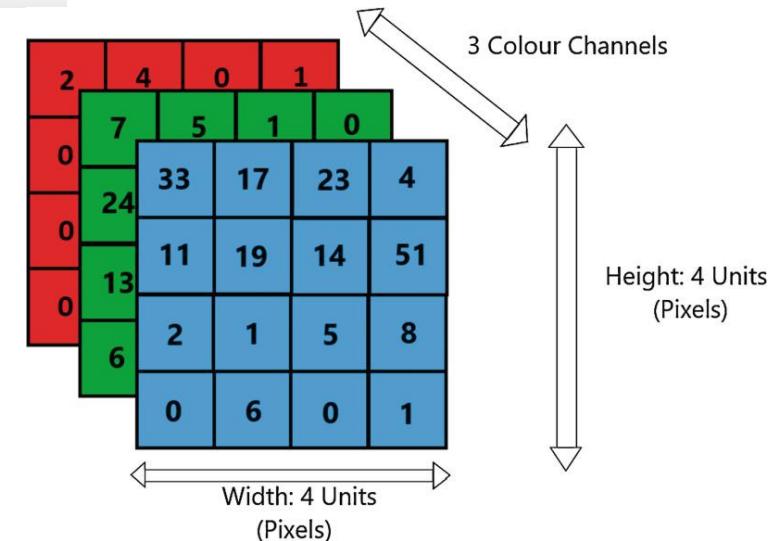
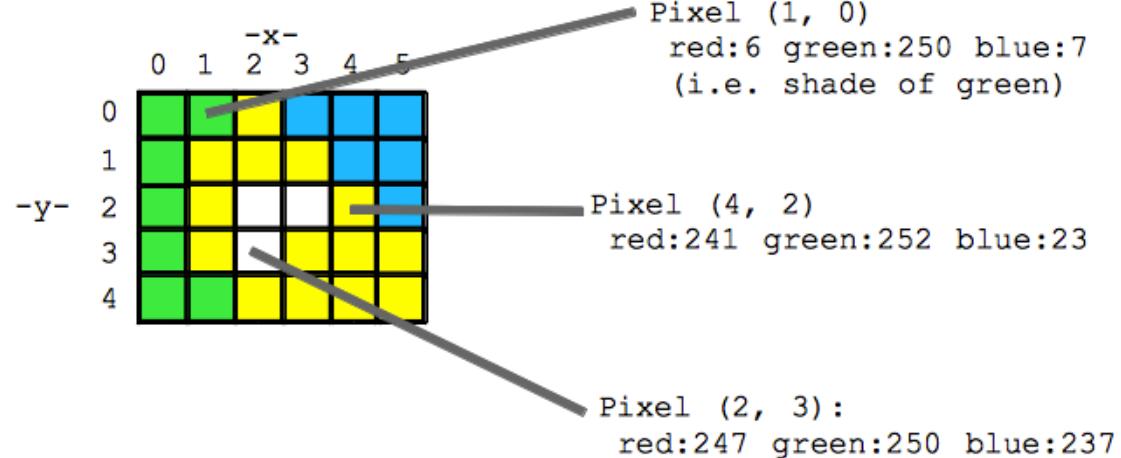


digital image fundamental pixel by pixel



| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 157 | 153 | 174 | 168 | 142 | 152 | 129 | 151 | 172 | 161 | 158 | 156 |
| 158 | 182 | 163 | 74 | 75 | 62 | 93 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 108 | 6 | 124 | 131 | 125 | 204 | 166 | 15 | 56 | 180 | |
| 194 | 68 | 127 | 251 | 237 | 299 | 239 | 228 | 227 | 47 | 71 | 201 |
| 172 | 106 | 297 | 233 | 230 | 214 | 220 | 239 | 238 | 98 | 74 | 206 |
| 188 | 88 | 179 | 208 | 188 | 215 | 311 | 168 | 138 | 79 | 20 | 159 |
| 189 | 97 | 168 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 193 | 193 | 188 | 227 | 178 | 149 | 162 | 96 | 36 | 190 |
| 205 | 174 | 158 | 252 | 236 | 231 | 149 | 179 | 238 | 43 | 95 | 234 |
| 190 | 214 | 116 | 149 | 236 | 187 | 95 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 158 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 68 | 109 | 143 | 96 | 60 | 2 | 198 | 249 | 216 |
| 187 | 196 | 238 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 258 | 211 |
| 180 | 209 | 237 | 149 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 129 | 207 | 177 | 131 | 123 | 206 | 179 | 13 | 96 | 218 |

| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 157 | 163 | 174 | 168 | 160 | 152 | 129 | 151 | 172 | 161 | 158 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 93 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 108 | 6 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 106 | 297 | 233 | 233 | 214 | 220 | 239 | 238 | 98 | 74 | 206 |
| 188 | 88 | 179 | 259 | 185 | 215 | 211 | 188 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 188 | 227 | 178 | 149 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 282 | 236 | 231 | 149 | 179 | 229 | 49 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 109 | 143 | 96 | 60 | 2 | 109 | 249 | 216 |
| 187 | 196 | 236 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 149 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 196 | 206 | 129 | 207 | 177 | 131 | 129 | 200 | 175 | 13 | 96 | 218 |



digital image fundamental pixel by pixel



| | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|---|
| 0 | 2 | 15 | 0 | 0 | 11 | 10 | 0 | 0 | 0 | 0 | 9 | 9 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 4 | 60 | 157 | 236 | 255 | 255 | 177 | 95 | 61 | 32 | 0 | 0 | 29 | |
| 0 | 10 | 16 | 119 | 238 | 255 | 244 | 245 | 243 | 250 | 249 | 255 | 222 | 103 | 10 | 0 | |
| 0 | 14 | 170 | 255 | 255 | 244 | 254 | 255 | 253 | 245 | 255 | 249 | 253 | 251 | 124 | 1 | |
| 2 | 98 | 255 | 228 | 255 | 251 | 254 | 211 | 141 | 116 | 122 | 215 | 251 | 238 | 255 | 49 | |
| 13 | 217 | 243 | 255 | 155 | 33 | 226 | 52 | 2 | 0 | 10 | 13 | 232 | 255 | 36 | | |
| 16 | 229 | 252 | 254 | 49 | 12 | 0 | 0 | 7 | 7 | 0 | 70 | 237 | 252 | 235 | 62 | |
| 6 | 141 | 245 | 255 | 212 | 25 | 11 | 9 | 3 | 0 | 115 | 236 | 243 | 255 | 137 | 0 | |
| 0 | 87 | 252 | 250 | 248 | 215 | 60 | 0 | 1 | 121 | 252 | 255 | 248 | 144 | 6 | 0 | |
| 0 | 13 | 113 | 255 | 255 | 245 | 255 | 182 | 181 | 248 | 252 | 242 | 208 | 36 | 0 | 19 | |
| 1 | 0 | 5 | 117 | 251 | 255 | 241 | 255 | 247 | 255 | 241 | 152 | 17 | 0 | 7 | 0 | |
| 0 | 0 | 0 | 0 | 4 | 58 | 251 | 255 | 246 | 254 | 253 | 255 | 120 | 11 | 0 | 1 | |
| 0 | 0 | 4 | 97 | 255 | 255 | 255 | 248 | 252 | 255 | 244 | 255 | 182 | 10 | 0 | 4 | |
| 0 | 22 | 206 | 252 | 246 | 251 | 241 | 100 | 24 | 113 | 255 | 245 | 255 | 194 | 9 | 0 | |
| 0 | 111 | 255 | 242 | 255 | 158 | 24 | 0 | 0 | 6 | 39 | 255 | 232 | 230 | 56 | 0 | |
| 0 | 218 | 251 | 250 | 137 | 7 | 11 | 0 | 0 | 0 | 2 | 62 | 255 | 250 | 125 | 3 | |
| 0 | 173 | 255 | 255 | 101 | 9 | 20 | 0 | 13 | 3 | 13 | 182 | 251 | 245 | 61 | 0 | |
| 0 | 107 | 251 | 241 | 255 | 230 | 98 | 55 | 19 | 118 | 217 | 248 | 253 | 255 | 52 | 4 | |
| 0 | 18 | 146 | 250 | 255 | 247 | 255 | 255 | 249 | 255 | 240 | 255 | 129 | 0 | 5 | | |
| 0 | 0 | 23 | 113 | 215 | 255 | 250 | 248 | 255 | 255 | 248 | 248 | 118 | 14 | 12 | 0 | |
| 0 | 0 | 6 | 1 | 0 | 52 | 153 | 233 | 255 | 252 | 147 | 37 | 0 | 0 | 4 | 1 | |
| 0 | 0 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 14 | 1 | 0 | 6 | 6 | 0 | 0 | |



```
0 2 15 0 0 11 10 0 0 0 0 9 9 0 0 0
0 0 0 4 60 157 236 255 255 177 95 61 32 0 0 29
0 10 16 119 238 255 244 245 243 250 249 255 222 103 10 0
0 14 170 255 255 244 254 255 253 245 255 249 253 251 124 1
2 98 255 228 255 251 254 211 141 116 122 215 251 238 255 49
13 217 243 255 155 33 226 52 2 0 10 13 232 255 36
16 229 252 254 49 12 0 0 7 7 0 70 237 252 235 62
6 141 245 255 212 25 11 9 3 0 115 236 243 255 137 0
0 87 252 250 248 215 60 0 1 121 252 255 248 144 6 0
0 13 113 255 255 245 255 182 181 248 252 242 208 36 0 19
1 0 5 117 251 255 241 255 247 255 241 152 17 0 7 0
0 0 0 4 58 251 255 246 254 253 255 120 11 0 1 0
0 0 4 97 255 255 255 248 252 255 244 255 182 10 0 4
0 22 206 252 246 251 241 100 24 113 255 245 255 194 9 0
0 111 255 242 255 158 24 0 0 6 39 255 232 230 56 0
0 218 251 250 137 7 11 0 0 0 2 62 255 250 125 3
0 173 255 255 101 9 20 0 13 3 13 182 251 245 61 0
0 107 251 241 255 230 98 55 19 118 217 248 253 255 52 4
0 18 146 250 255 247 255 255 249 255 240 255 129 0 5
0 0 23 113 215 255 250 248 255 255 248 248 118 14 12 0
0 0 6 1 0 52 153 233 255 252 147 37 0 0 4 1
0 0 5 5 0 0 0 0 0 14 1 0 6 6 0 0
```

Image processing on DL(Mnist)

```
[2] import tensorflow as tf
    from tensorflow import keras
    import matplotlib.pyplot as plt
    import numpy as np

[4] mnist=keras.datasets.mnist

[5] (X_train,Y_train),(X_test,Y_test)=mnist.load_data()

[37] plt.imshow(X_train[10])
    Y_train[10]
```

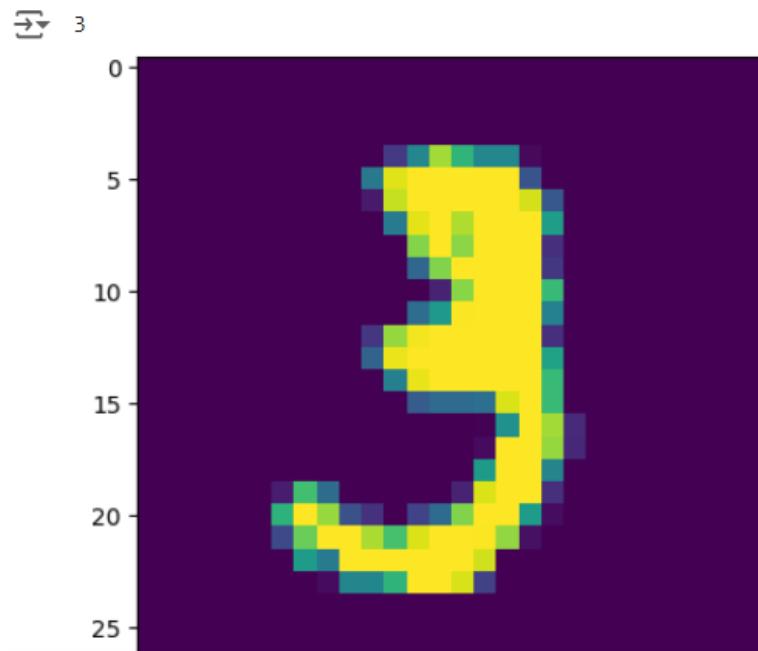


Image processing on DL(Mnist)

```
▶ model=keras.Sequential()  
model.add(keras.layers.Flatten(input_shape=(28,28)))  
model.add(keras.layers.Dense(128,activation='sigmoid'))  
model.add(keras.layers.Dense(256,activation='sigmoid'))  
model.add(keras.layers.Dense(10,activation='softmax'))
```

```
[39] model.compile(optimizer='adam',loss="sparse_categorical_crossentropy")
```

```
[40] model.summary()
```

→ Model: "sequential_7"

| Layer (type) | Output Shape | Param # |
|---------------------|--------------|---------|
| flatten_7 (Flatten) | (None, 784) | 0 |
| dense_15 (Dense) | (None, 128) | 100,480 |
| dense_16 (Dense) | (None, 256) | 33,024 |
| dense_17 (Dense) | (None, 10) | 2,570 |

Total params: 136,074 (531.54 KB)

Trainable params: 136,074 (531.54 KB)

Non-trainable params: 0 (0.00 B)

Image processing on DL(Mnist)

```
[41] model.fit(X_train,Y_train,batch_size=600,epochs=200)
```

```
[42] Y_predict_train=model.predict(X_train)  
Y_predict_test=model.predict(X_test)
```

```
→ 1875/1875 ━━━━━━━━ 4s 2ms/step  
313/313 ━━━━━━ 1s 2ms/step
```

```
[43] print(Y_train[10])  
Y_predict_train[10]
```

```
→ 3  
array([1.3138961e-07, 1.4539218e-07, 7.4563314e-07, 9.9993950e-01,  
      3.8355483e-09, 1.3472881e-05, 1.0117575e-11, 2.4469269e-08,  
      1.8759265e-05, 2.7276197e-05], dtype=float32)
```

```
▶ print(Y_test[10])  
Y_predict_test[10]
```

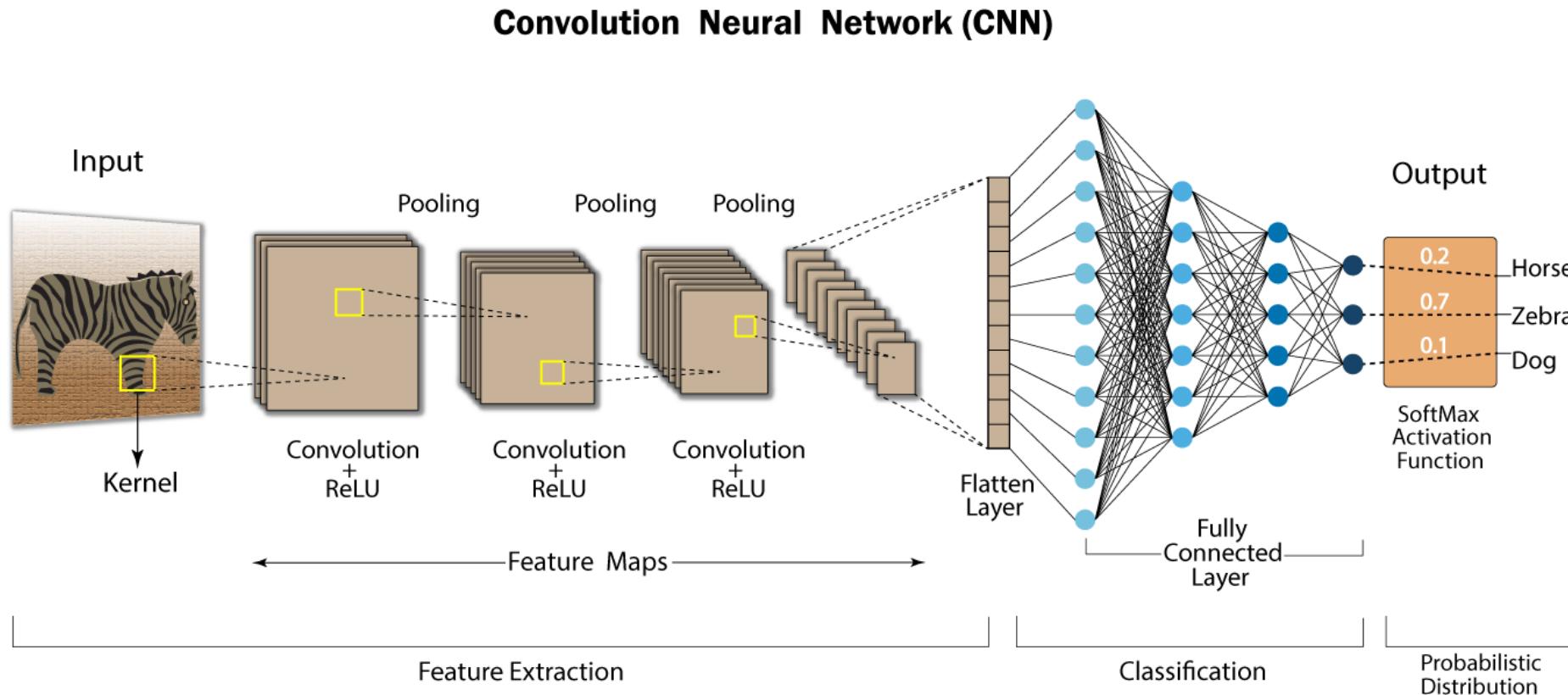
```
→ 0  
array([9.9193335e-01, 6.4419521e-07, 1.5828909e-05, 1.0436028e-06,  
      6.8988920e-05, 5.9089061e-06, 1.7361279e-05, 5.3363374e-06,  
      5.6906168e-07, 7.9509579e-03], dtype=float32)
```

Image processing on DL(Mnist)

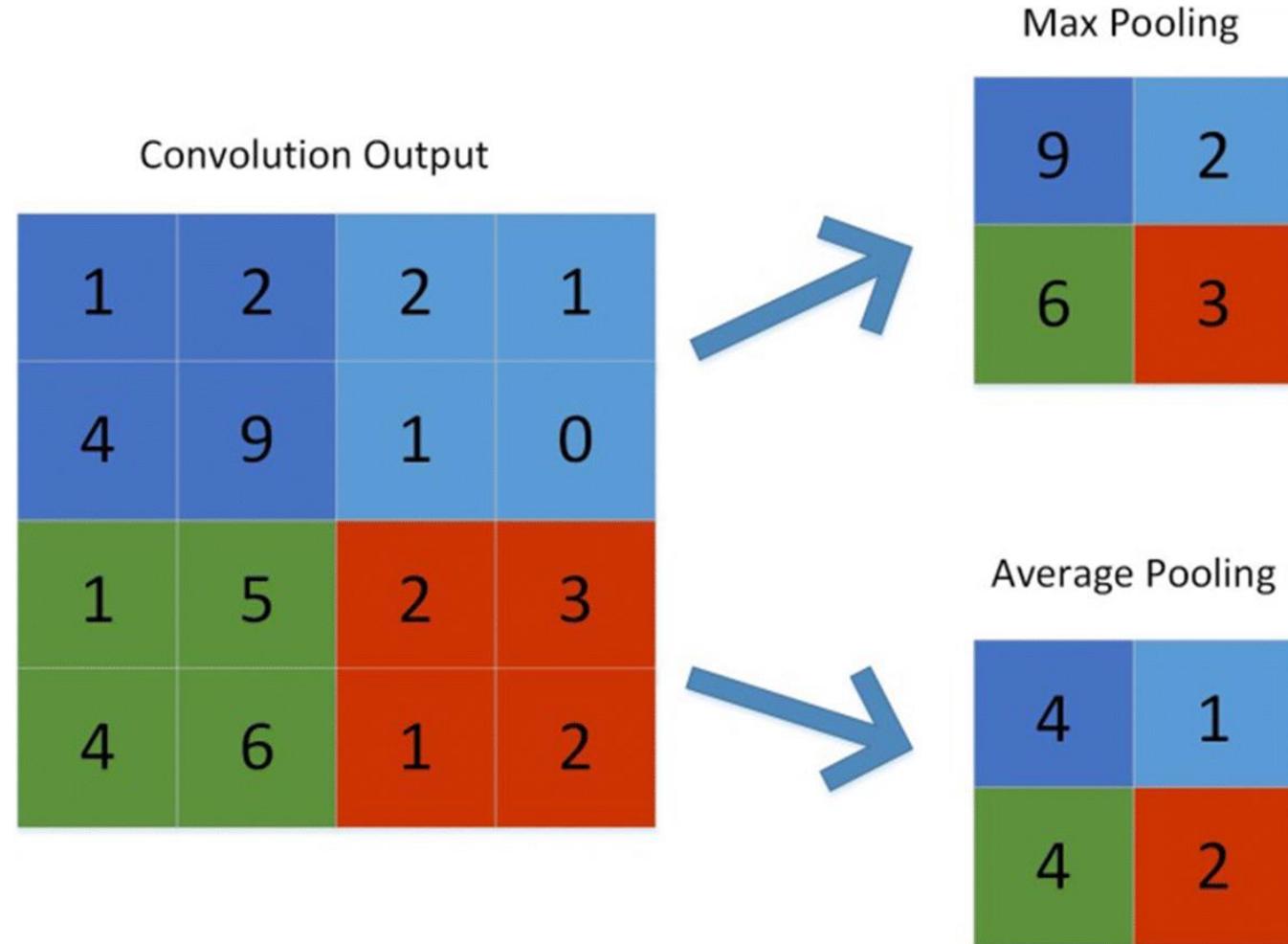
```
[57] test_data = np.reshape(X_train[10], (1, 28, 28))
      test_data.shape
→ (1, 28, 28)

▶ model.predict(test_data)
→ 1/1 ━━━━━━━━ 0s 30ms/step
array([[1.31388731e-07, 1.45396626e-07, 7.45621094e-07, 9.99939561e-01,
       3.83560739e-09, 1.34730235e-05, 1.01174798e-11, 2.44694558e-08,
       1.87594451e-05, 2.72769012e-05]], dtype=float32)
```

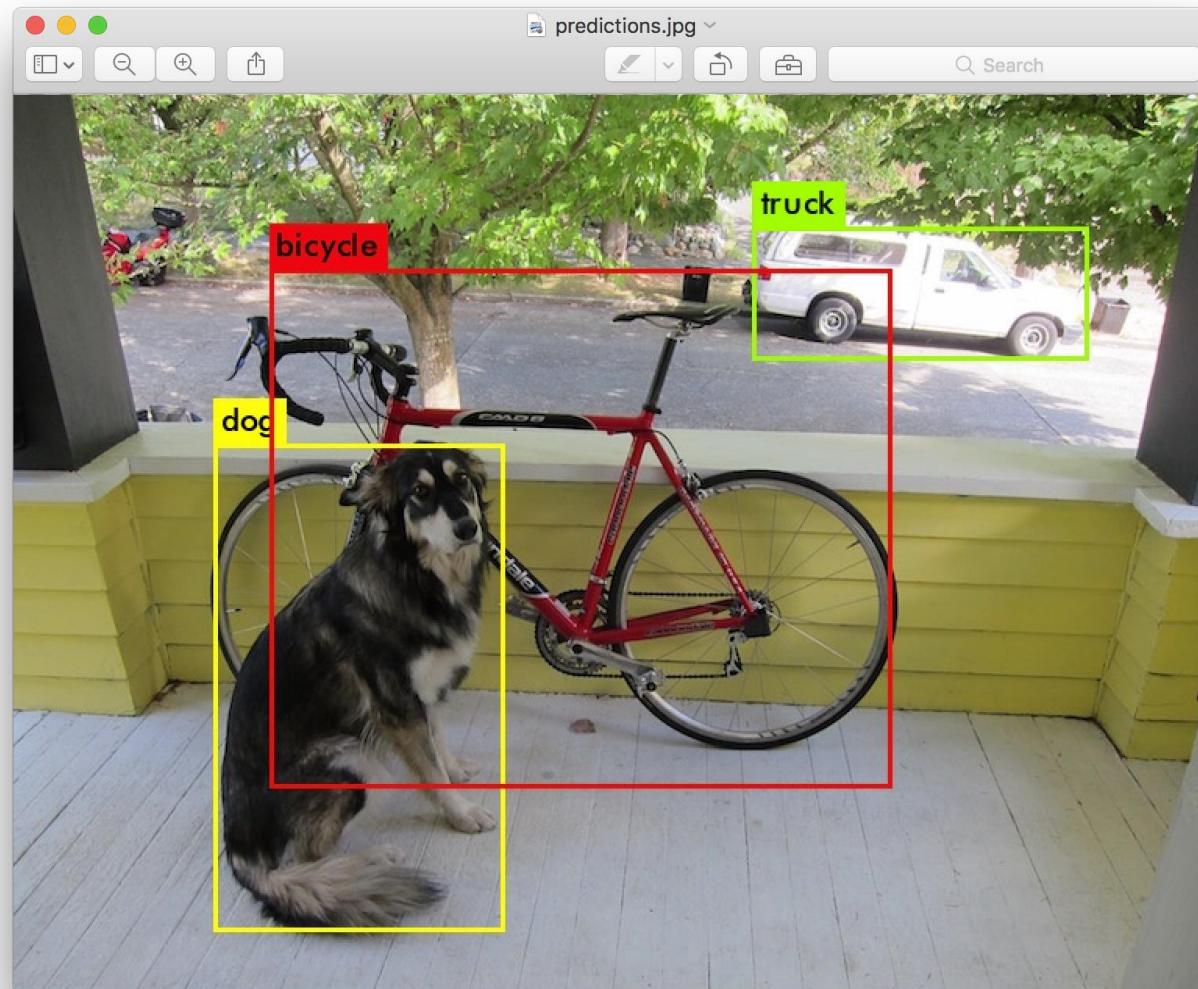
Convolutional Neural Network (CNN)



pooling



YOLO



conclusion

Thank you for your support



Sin.majidi@gmail.com