MATLOBLIB AND INTRO TO MACHINE LEARNING

What is Matplotlib

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Matplotlib was created by John D. Hunter.
- Matplotlib is open source and we can use it freely.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.
- The source code for Matplotlib is located at this github repository https://github.com/matplotlib/matplotlib

PYPLOT

- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias.

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

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

Plotting x and y points

- The plot() function is used to draw points (markers) in a diagram.
- By default, the plot() function draws a line from point to point.
- The function takes parameters for specifying points in the diagram.
- Parameter 1 is an array containing the points on the **x-axis**.
- Parameter 2 is an array containing the points on the **y-axis**.
- If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

Plotting Without Line

- To plot only the markers, you can use *shortcut string notation* parameter 'o', which means 'rings'.

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

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```

Multiple Points

- You can plot as many points as you like, just make sure you have the same number of points in both axis.

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

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

MATPLOTLIB MARKERS

- You can use the keyword argument marker to emphasize each point with a specified marker.

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```

- plt.plot(ypoints, marker = '*').

MARKER REFERENCE

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)
'p'	Pentagon
'H'	Hexagon

Hexagon
Triangle Down
Triangle Up
Triangle Left
Triangle Right
Tri Down
Tri Up
Tri Left
Tri Right
Vline
Hline

CONTINUED

Format Strings fmt

- You can use also use the *shortcut string notation* parameter to specify the marker.
- This parameter is also called fmt, and is written with this syntax.

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'o:r')
plt.show()
```

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line

LINE REFERENCE

COLOR REFRENCE

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Marker Size

- You can use the keyword argument markersize or the shorter version, ms to set the size of the markers.

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20)
plt.show()
```

Marker Color

- You can use the keyword argument markeredgecolor or the shorter mec to set the color of the *edge* of the markers.

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')
plt.show()
```

CONTINUED

- YOU CAN USE THE KEYWORD ARGUMENT MARKERFACECOLOR OR THE SHORTER MFC TO SET THE COLOR INSIDE THE EDGE OF THE MARKERS.

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

YPOINTS = NP.ARRAY([3, 8, 1, 10])

PLT.PLOT(YPOINTS, MARKER = 'O', MS = 20, MFC = 'R')
PLT.SHOW()
```

CONTINUED

- USE *BOTH* THE MEC AND MFC ARGUMENTS TO COLOR OF THE ENTIRE MARKER.

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

YPOINTS = NP.ARRAY([3, 8, 1, 10])

PLT.PLOT(YPOINTS, MARKER = 'O', MS = 20, MEC = 'R', MFC = 'R')
PLT.SHOW()
```

Matplotlib Line

- You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line.

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```

- plt.plot(ypoints, linestyle = 'dashed')

Line Width

- You can use the keyword argument `linewidth` or the shorter `lw` to change the width of the line.

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linewidth = '20.5')
plt.show()
```

Multiple Lines

- Draw two lines by specifying a plt.plot() function for each line.

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

y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)

plt.show()
```

Matplotlib Labels and Title

- With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```

Create a Title for a Plot

- With Pyplot, you can use the `title()` function to set a title for the plot.

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```

Position the Title

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data", loc = 'left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
plt.show()
```

Matplotlib Adding Grid Lines

- With Pyplot, you can use the grid() function to add grid lines to the plot.

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```

Specify which grid lines to display

- You can use the axis parameter in the grid() function to specify which grid lines to display.
- Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(axis = 'x')

plt.show()
```

SET LINE PROPERTIES FOR THE GRID

- You can also set the line properties of the grid, like this: grid(color = '*color*', linestyle = '*linestyle*', linewidth = *number*).

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y =
np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)

plt.show()
```

MATPLOTLIB SUBPLOTS

- WITH THE **SUBPLOTS()** FUNCTION YOU CAN DRAW MULTIPLE PLOTS IN ONE FIGURE.

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

#PLOT 1:
X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([3, 8, 1, 10])

PLT.SUBPLOT(1, 2, 1)
PLT.PLOT(X,Y)

#PLOT 2:
X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([10, 20, 30, 40])

PLT.SUBPLOT(1, 2, 2)
PLT.PLOT(X,Y)

PLT.SHOW()
```

The subplots() function

- The subplots( ) function takes three arguments that describes the layout of the figure.
- The layout is organized in rows and columns, which are represented by the *first* and *second* argument.
- The third argument represents the index of the current plot.

```
plt.subplot(1, 2, 1)
```

ON TOP OF EACH OTHER

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

#PLOT 1:
X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([3, 8, 1, 10])

PLT.SUBPLOT(2, 1, 1)
PLT.PLOT(X,Y)

#PLOT 2:
X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([10, 20, 30, 40])

PLT.SUBPLOT(2, 1, 2)
PLT.PLOT(X,Y)

PLT.SHOW()
```

DRAW SIX PLOTS

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([3, 8, 1, 10])

PLT.SUBPLOT(2, 3, 1)
PLT.PLOT(X,Y)

X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([10, 20, 30, 40])

PLT.SUBPLOT(2, 3, 2)
PLT.PLOT(X,Y)

X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([3, 8, 1, 10])

PLT.SUBPLOT(2, 3, 3)
PLT.PLOT(X,Y)

X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([10, 20, 30, 40])

PLT.SUBPLOT(2, 3, 4)
PLT.PLOT(X,Y)

X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([3, 8, 1, 10])

PLT.SUBPLOT(2, 3, 5)
PLT.PLOT(X,Y)

X = NP.ARRAY([0, 1, 2, 3])
Y = NP.ARRAY([10, 20, 30, 40])

PLT.SUBPLOT(2, 3, 6)
PLT.PLOT(X,Y)

PLT.SHOW ()
```

GIVING TITLES TO MULTIPLE PLOTS

- You can add a title to each plot with the `title()` function.

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

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.show()
```

SUPER TITLE

- You can add a title to the entire figure with the `suptitle()` function.

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

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```

SCATTER PLOTS

- With Pyplot, you can use the scatter() function to draw a scatter plot.
- The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

EXAMPLE

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

X = NP.ARRAY([5,7,8,7,2,17,2,9,4,11,12,9,6])
Y = NP.ARRAY([99,86,87,88,111,86,103,87,94,78,77,85,86])

PLT.SCATTER(X, Y)
PLT.SHOW()
```

Compare Plots

- In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

EXAMPLE

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

#DAY ONE, THE AGE AND SPEED OF 13 CARS:
X = NP.ARRAY([5,7,8,7,2,17,2,9,4,11,12,9,6])
Y = NP.ARRAY([99,86,87,88,111,86,103,87,94,78,77,85,86])
PLT.SCATTER(X, Y)

#DAY TWO, THE AGE AND SPEED OF 15 CARS:
X = NP.ARRAY([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
Y = NP.ARRAY([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
PLT.SCATTER(X, Y)

PLT.SHOW()
```

ALPHA

- You can adjust the transparency of the dots with the **alpha** argument.
- Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

EXAMPLE

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)

plt.show()
```

COMBINE COLOR SIZE AND ALPHA

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

x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()

plt.show()
```

Matplotlib Bars

With Pyplot, you can use the bar() function to draw bar graphs.

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

Horizontal Bars

- If you want the bars to be displayed horizontally instead of vertically, use the barh() function.

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```

Bar Width

- The `bar()` takes the keyword argument `width` to set the width of the bars.

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)
plt.show()
```

BAR HEIGHT

- The barh( ) takes the keyword argument height to set the height of the bars.

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```

Matplotlib Histograms

- A histogram is a graph showing *frequency* distributions.
- It is a graph showing the number of observations within each given interval.
- Example: Say you ask for the height of 250 people, you might end up with a histogram like this.
- In Matplotlib, we use the `hist()` function to create histograms.
- The `hist()` function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

EXAMPLE

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

X = NP.RANDOM.NORMAL(170, 10, 250)

PLT.HIST(X)
PLT.SHOW()
```

Matplotlib Pie Charts

- With Pyplot, you can use the `pie()` function to draw pie charts.

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

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```

Explode

- Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.
- The explode parameter, if specified, and not None, must be an array with one value for each wedge.
- Each value represents how far from the center each wedge is displayed.

EXAMPLE

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

  y = np.array([35, 25, 25, 15])
  mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
  myexplode = [0.2, 0, 0, 0]

  plt.pie(y, labels = mylabels, explode = myexplode)
  plt.show()
  ```
  ```

Legend With Header

- To add a header to the legend, add the `title` parameter to the `legend` function

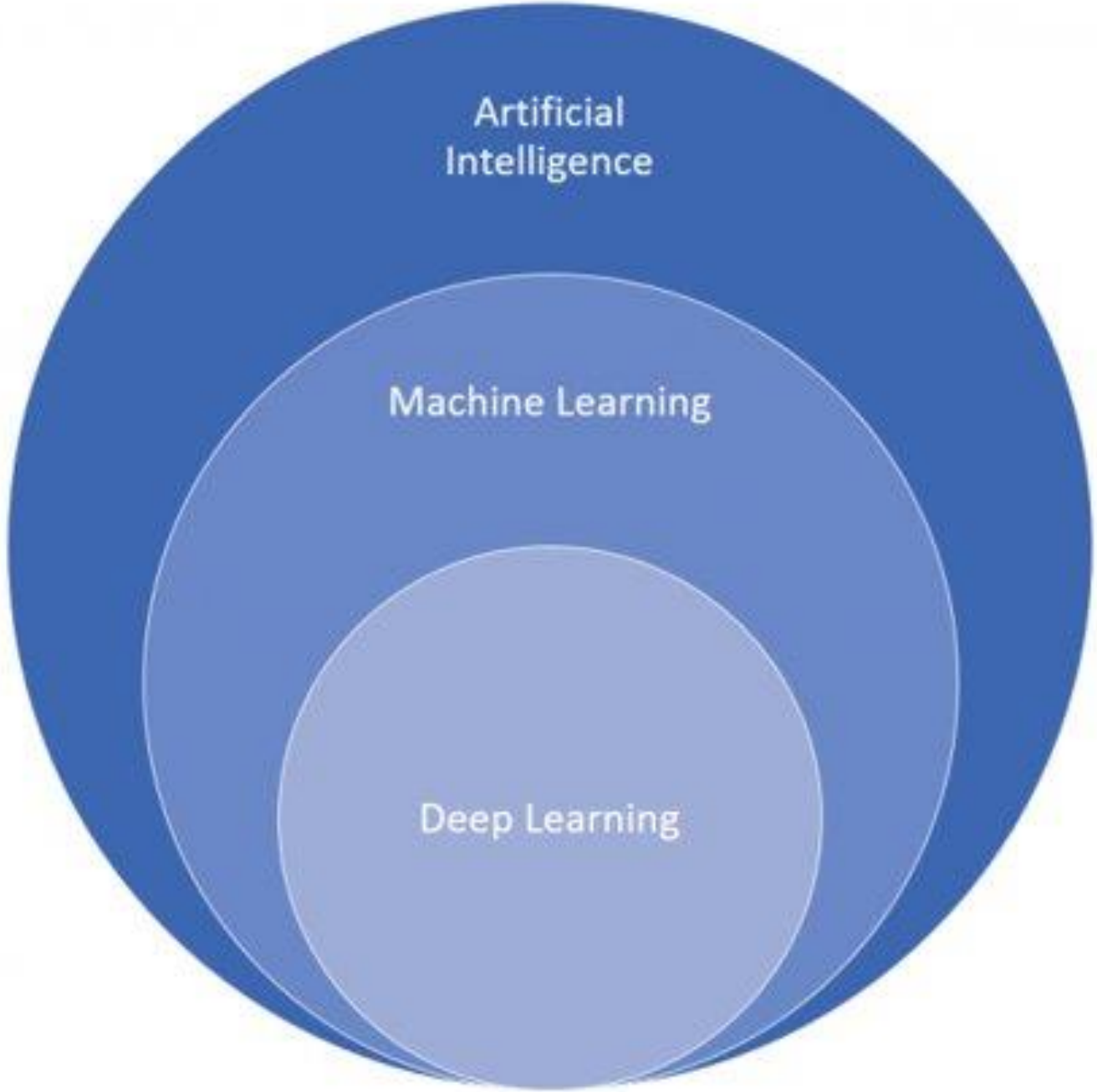
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```

Artificial Intelligence vs Machine Learning vs Deep Learning

- Artificial Intellgence: Simulating the human brains work , interactions , perceptions , emotions etc.
- Machine Learnning: A subset of AI and is a means to achieve AI.
- Deep Learning: A subset of machine learning and a powerful technique for achieve AI.


Artificial
Intelligence
Machine Learning
Deep Learning

Structured data vs unstructured data

- Structured : numbers, dates , value .. etc and can be stored in rows and columns.
- Unstructured : text, audio, video … etc stored in noSQL databases.

Learning types

- Supervised Learning: where the data is labeled. (classification and regression)
- Unsupervised Learning: analyze and cluster unlabeled data, it discovers patterns in data without the need for human intervention. (Clusteriung , Association, Dimensionality reduction).
- Reinforcement Learning: reward punishment system. (usually used in gaming)

Supervised learning

- Classification: Accurately assigns data to specific categories . The most common example is separating spam from not spam. (Linear classifiers, SVMs, decision trees, logistic regression)
- Regression: Predicting numerical values based on different data points. Most common examples is housing prices. (linear regression, polynomial regression).

Unsupervised learning

- Clusteruing: grouping unlabeled data based on their similarities or differences, (K-means).
- Association: measuring the degress of association between two items.
- Dimensionality Reduction: used when the number of features is too high . It reduces the number of inputs to a manageable size size while also pers