Topic 3 Lecture - Introduction to Computer Vision and Machine Learning

Aims of the Session

- Learn the basics on how images are imported to our computers and converted into data

Resources for the Lecture

- Introduction to Computing and Programming in Python: A Multimedia Approach. Mark Guzdial, Barbara Ericson. Pearson, 2016.
- Various others mentioned throughout the lecture!

How does an image look *digitally*?

How does an image look *digitally*?

- These are the main **compression algorithms** used to store images:

How does an image look *digitally*?

- These are the main **compression algorithms** used to store images:

Compression Algorithms

Compression Algorithms

- The *art* of compression algorithms is in **quantisation**!

Compression Algorithms

- The *art* of compression algorithms is in **quantisation**!
- Best algorithms are the ones that achieve best visual quality with reduced size.

Compression Algorithms

- The *art* of compression algorithms is in **quantisation**!
- Best algorithms are the ones that achieve best visual quality with reduced size.

8.9M

68.34K

- The importance of compression

- The importance of compression

In [1]:

```
import warnings
warnings.filterwarnings('ignore')
from IPython.display import HTML
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/NMk
```

Out[1]:

- Fortunately, this is **NOT** our problem in this module!

- Fortunately, this is **NOT** our problem in this module!
- We are going to work with images in *simpler* ways

Images as arrays/matrices

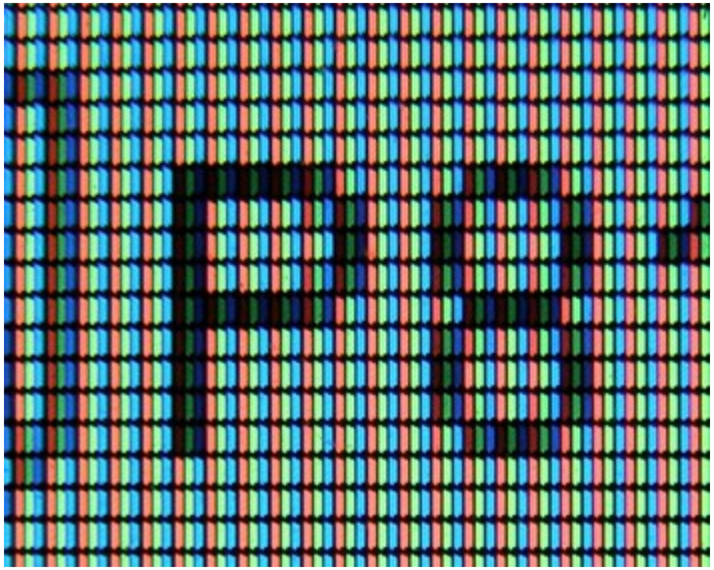
- Using the `numpy` module

Images as arrays/matrices

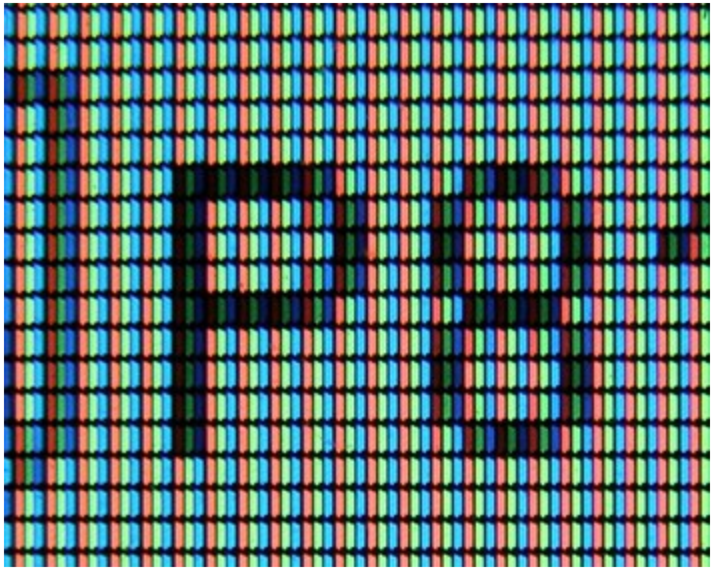
- Using the `numpy` module
- Complementing by using the `OpenCV` module, which will let us import and manipulate images

- When we import an image, the first thing we will get is a **bitmap**

- When we import an image, the first thing we will get is a **bitmap**



- When we import an image, the first thing we will get is a **bitmap**



- Each pixel will be represented as a value within an $n \times m$ matrix

Grayscale Images

Grayscale Images

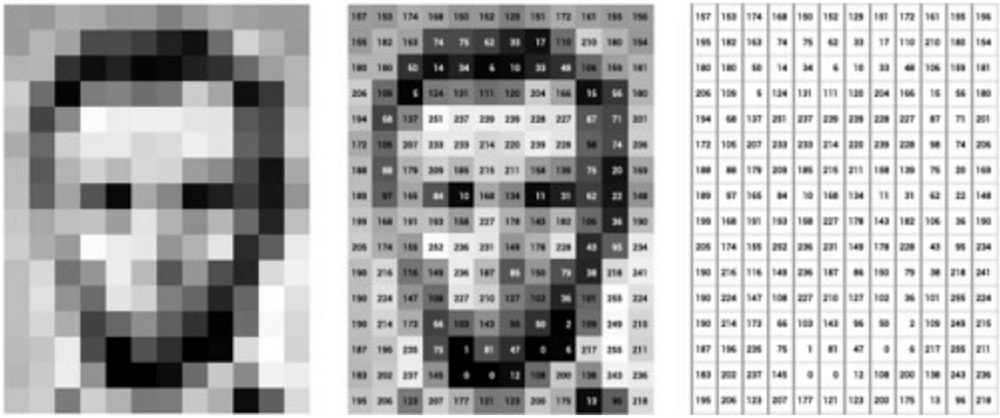
- A 2D grid of pixels

Grayscale Images

- A 2D grid of pixels
- Two ways to represent them:

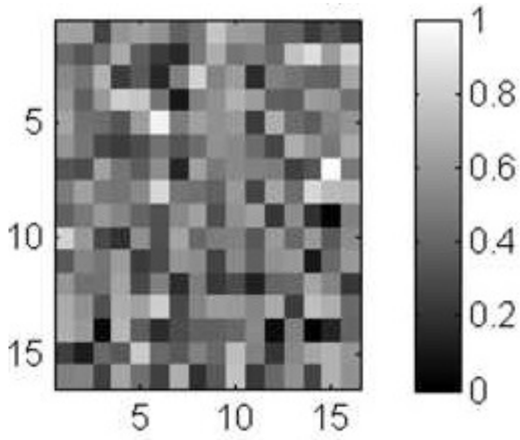
1. Standard: from 0 (black) to 255 (white) with 254 gray values in between.

1. Standard: from 0 (black) to 255 (white) with 254 gray values in between.



1. Normalised: from 0 (black) to 1 (white) with "infinite" gray values in between.

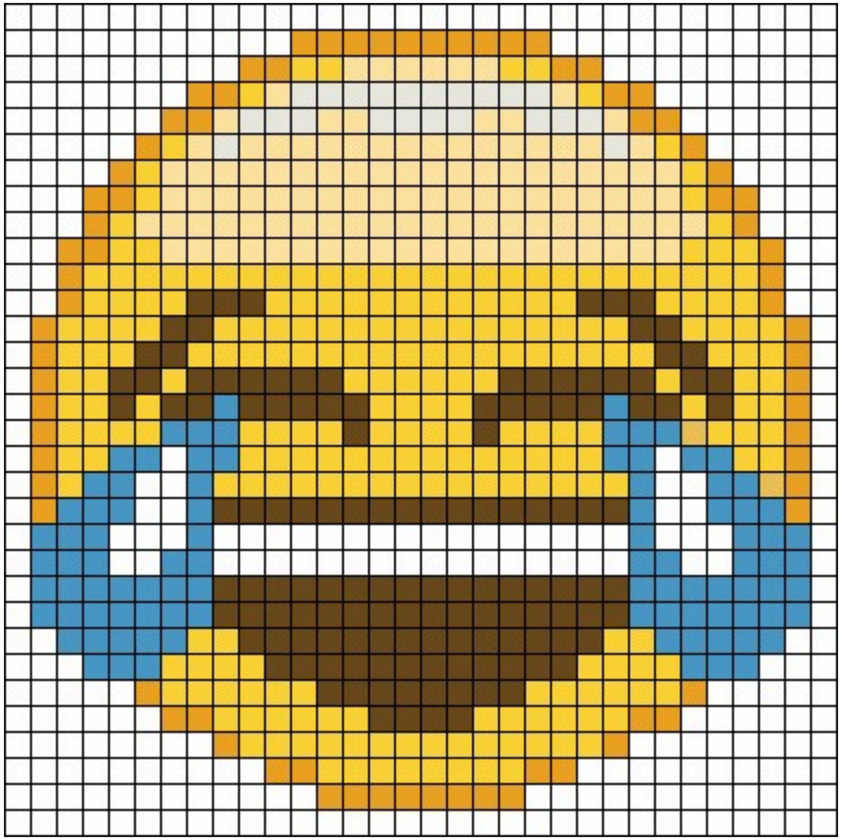
1. Normalised: from 0 (black) to 1 (white) with "infinite" gray values in between.



IS IT POSSIBLE TO CONVERT BETWEEN STANDARD $\leftrightarrow$ NORMALISED?

Colour Images

Colour Images



- Each pixel has three `channels` : $\color{red}{red}$, $\color{green}{green}$ and $\color{blue}{blue}$

- Each pixel has three `channels` : red, $green$ and $blue$
- Images with colour are often called RGB images

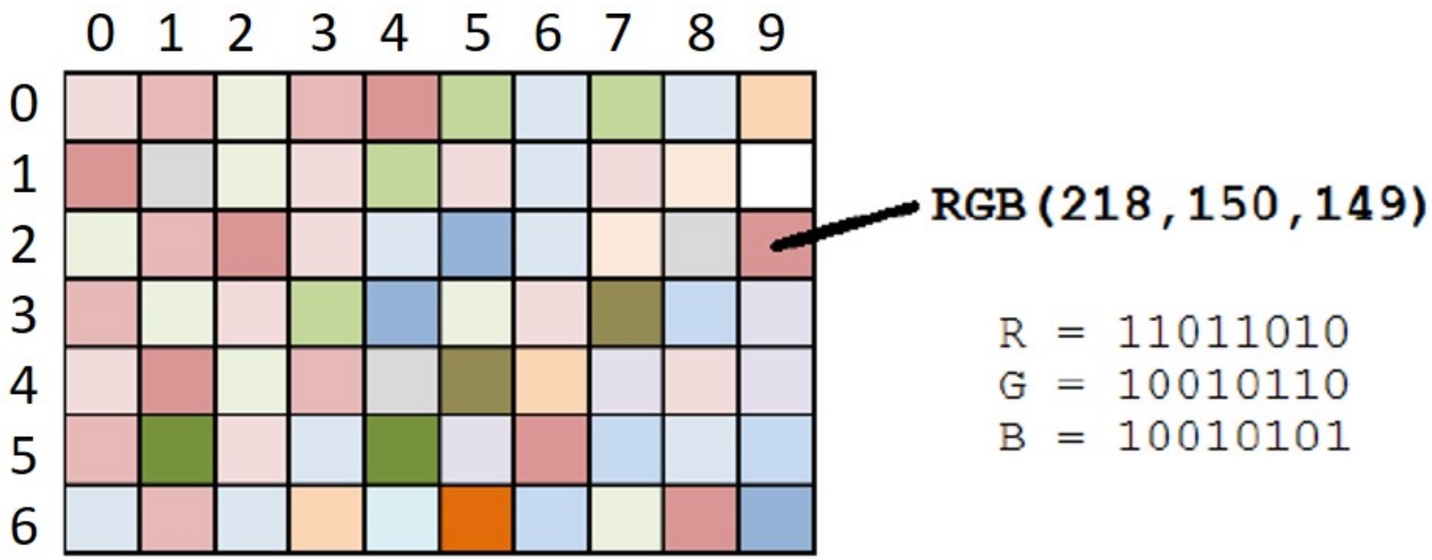
Option 1

Option 1

- If a colour image is imported, a matrix will be produced, this time with three values per pixel instead of one

Option 1

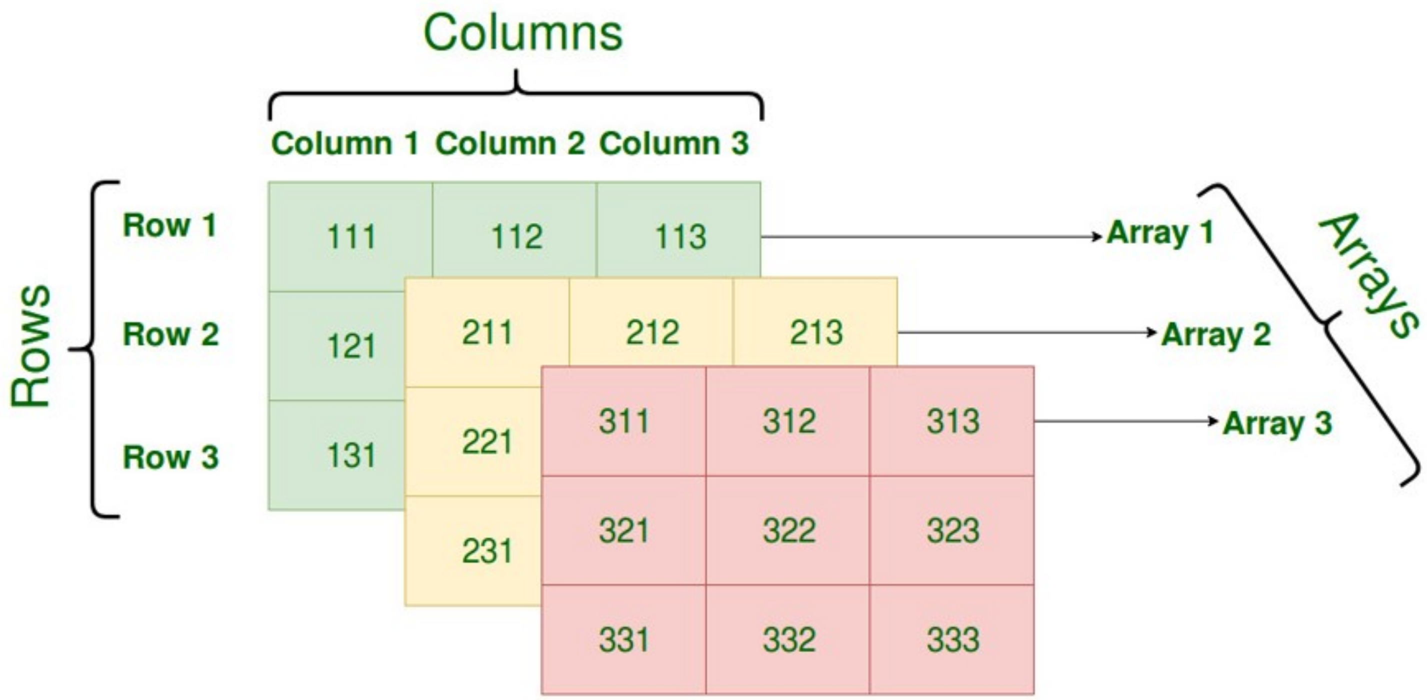
- If a colour image is imported, a matrix will be produced, this time with three values per pixel instead of one
- The three values will be stored in a tuple

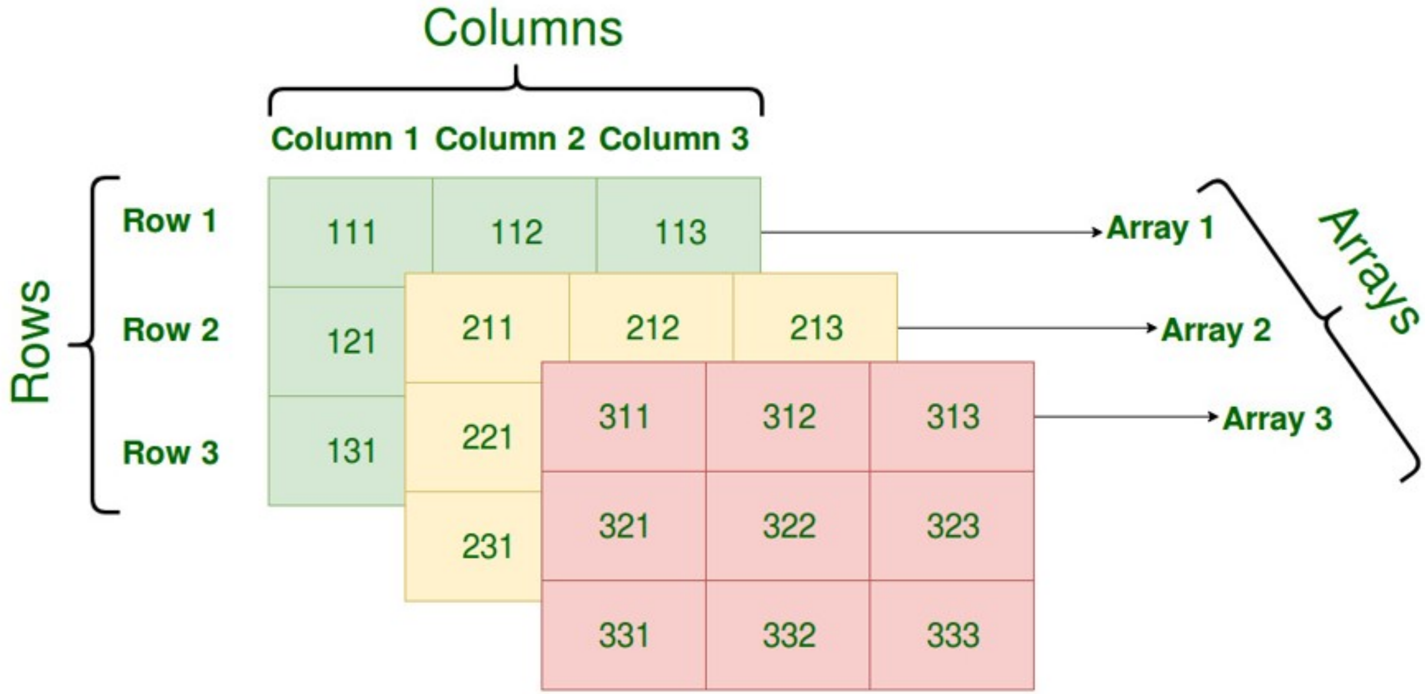

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
RGB(218,150,149)
R = 11011010
G = 10010110
B = 10010101

Option 2

Option 2

- When importing a colour image in `OpenCV`, a 3D array will be produced, with the third dimension representing the three channels


Columns
Column 1
Column 2
Column 3
Rows
Row 1
Row 2
Row 3
111
112
113
121
131
211
212
213
221
231
311
312
313
321
322
323
331
332
333
Array 1
Array 2
Array 3
Arrays



- Advantage of option 2: Faster to do calculations and transformations

HOW MANY COLOURS CAN BE REPRESENTED USING THIS STANDARD?

HOW MANY COLOURS CAN BE REPRESENTED USING THIS STANDARD?

CAN RGB BE NORMALISED?

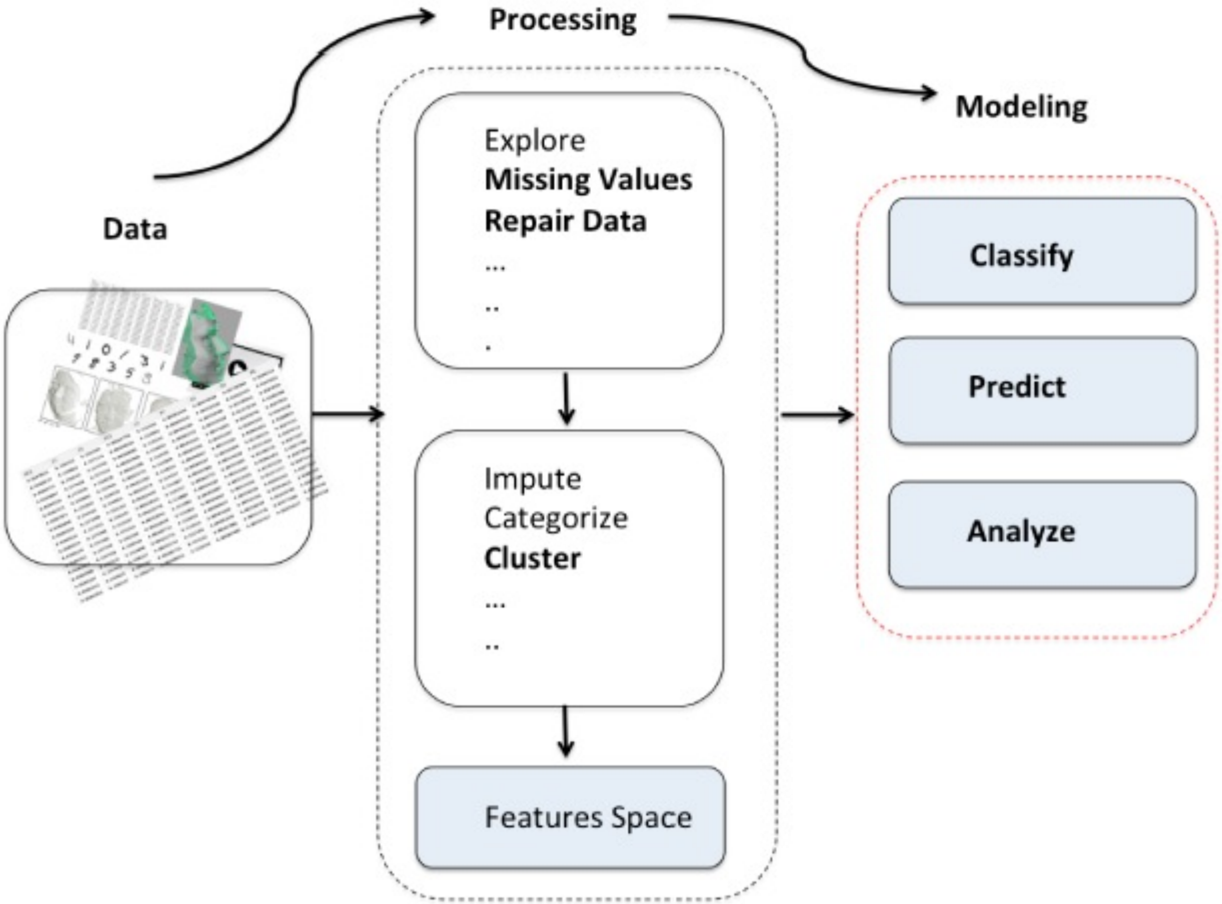
HOW MANY COLOURS CAN BE REPRESENTED USING THIS STANDARD?

CAN RGB BE NORMALISED?

ARE THERE ANY OTHER STANDARDS THAT CAN REPRESENT MORE COLOURS?

Machine Learning

- Machine learning: An automatic function that maps $x \rightarrow y$ based on the input data


Processing
Modeling
Data
Explore
Missing Values
Repair Data
...
..
.
Impute
Categorize
Cluster
...
..
Features Space
Classify
Predict
Analyze

Types of Machine Learning

Types of Machine Learning

Supervised Learning

Types of Machine Learning

Supervised Learning

- Aims to learn a function that, given a sample of data and desired outputs, approximates a function that maps inputs to output

Types of Machine Learning

Supervised Learning

- Aims to learn a function that, given a sample of data and desired outputs, approximates a function that maps inputs to output
- Done in the context of **classification** (when mapping input to output label) or **regression** (when mapping input to continuous output)

Types of Machine Learning

Supervised Learning

- Aims to learn a function that, given a sample of data and desired outputs, approximates a function that maps inputs to output
- Done in the context of **classification** (when mapping input to output label) or **regression** (when mapping input to continuous output)
- The "correct" output will be deduced from the `training` data, therefore the model requires a reliable base

Examples of Supervised Learning Algorithms

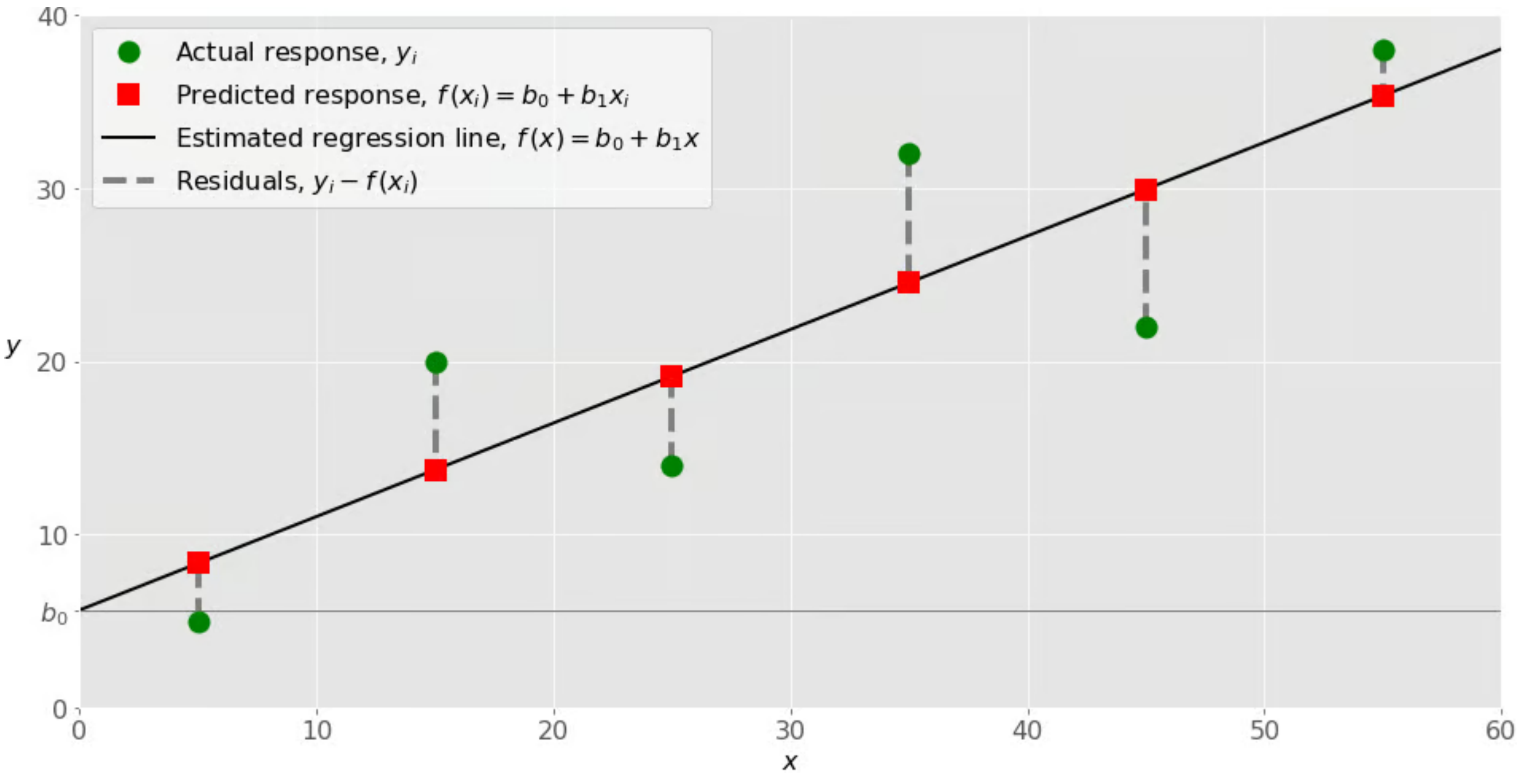
Regression

Regression

- The simplest ML out there! A line (or curve) that adapts to the data points and tries to do a prediction based on the existing data and how it adapts to a mathematical function

Regression

- The simplest ML out there! A line (or curve) that adapts to the data points and tries to do a prediction based on the existing data and how it adapts to a mathematical function



Source

- There's a version called logistic regression which uses a probability curve, which in turn can be adapted for binary classification!

- There's a version called logistic regression which uses a probability curve, which in turn can be adapted for binary classification!

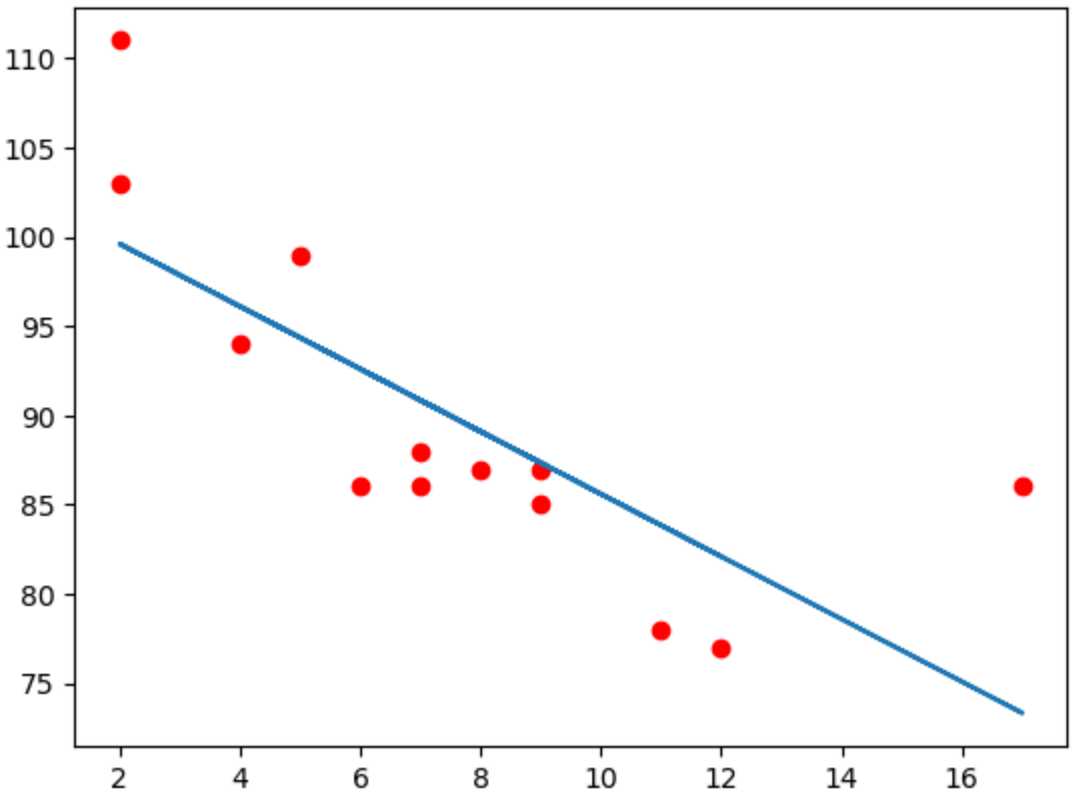
Source

- Also very simple code to implement

- Also very simple code to implement

In [2]:

```
# Source: https://www.w3schools.com/python/python_ml_linear_regression.asp

import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

# This is the lin reg, one line of code!
slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y, color='red')
plt.plot(x, mymodel)
plt.show()
```



110
105
100
95
90
85
80
75
2
4
6
8
10
12
14
16

K Nearest Neighbours

K Nearest Neighbours

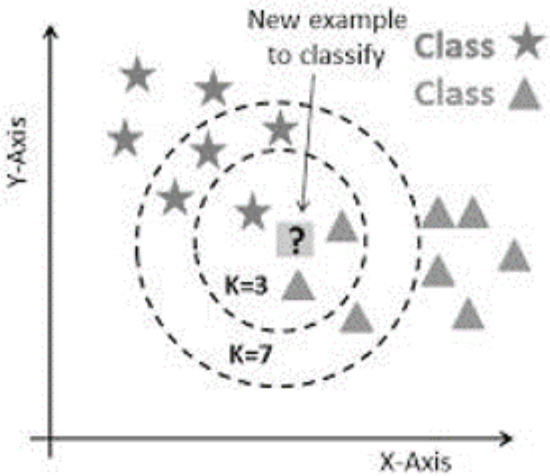
- Classify by a majority vote of neighbours

K Nearest Neighbours

- Classify by a majority vote of neighbours
- Advantages: Simple to implement, robust to noisy training data, and effective if training data is large

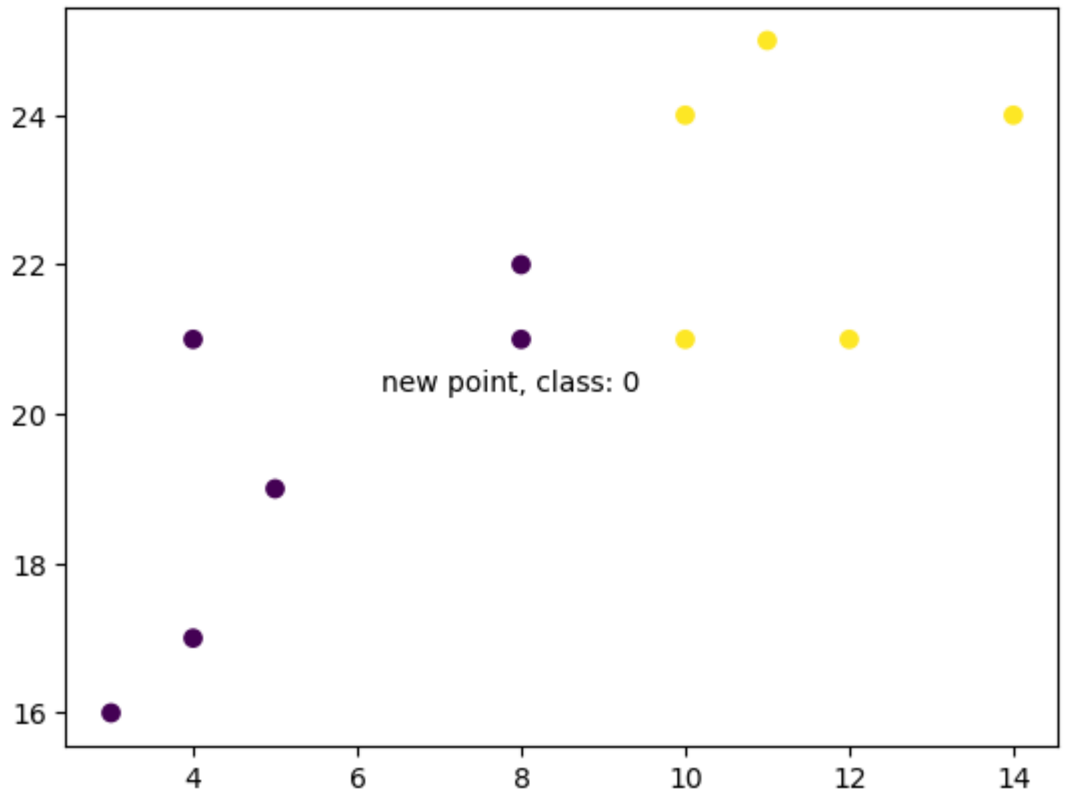
K Nearest Neighbours

- Classify by a majority vote of neighbours
- Advantages: Simple to implement, robust to noisy training data, and effective if training data is large
- Disadvantages: Need to determine the value of K computation cost is high


New example
to classify
Class
Class
Y-Axis
K=3
K=7
X-Axis

In [3]:

```python
# Source: https://www.w3schools.com/python/python_ml_knn.asp

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

x = [4, 5, 10, 4, 3, 11, 14 , 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
data = list(zip(x, y))

# Two lines of code to declare and "fit" the model!
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(data, classes)

# Test it
new_x = 8
new_y = 21

new_point = [(new_x, new_y)]
prediction = knn.predict(new_point)
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```


new point, class: 0
24
22
20
18
16
4
6
8
10
12
14

Decision Tree/Random Forest

Decision Tree/Random Forest

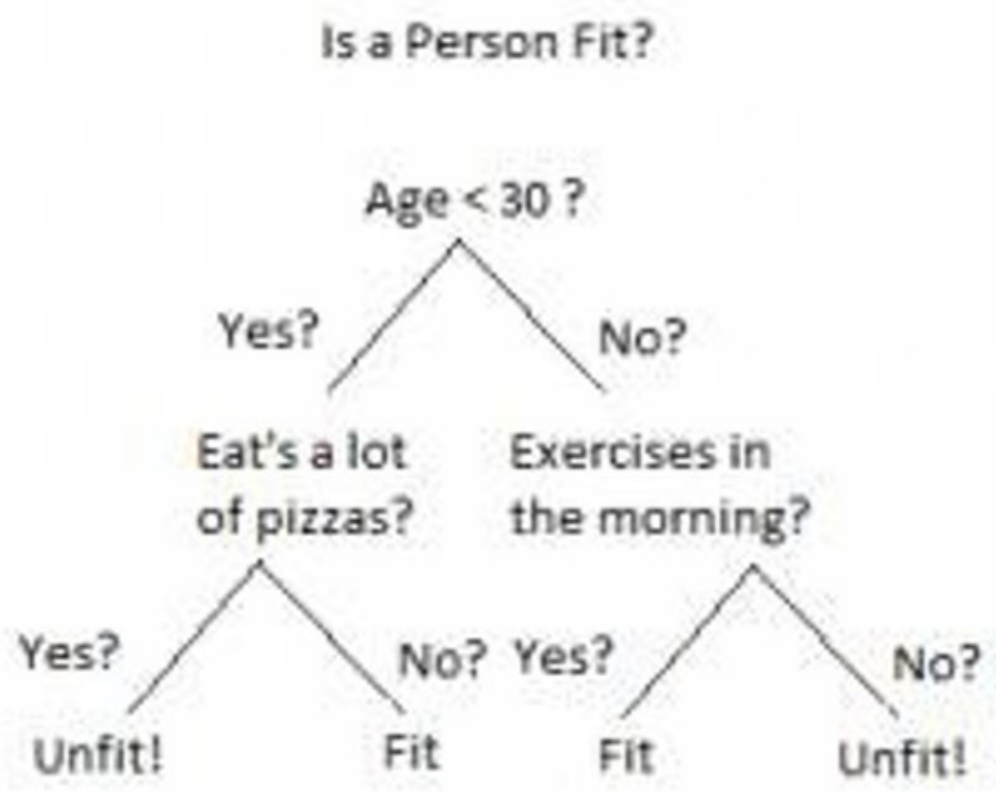
- Segment the predictor space into multiple regions

Decision Tree/Random Forest

- Segment the predictor space into multiple regions
- Each region has only a subset of the training dataset

Decision Tree/Random Forest

- Segment the predictor space into multiple regions
- Each region has only a subset of the training dataset
- High variance $\rightarrow$ Small changes in the training data can give an entirely different decision tree model


Is a Person Fit?
Age < 30 ?
Yes?
No?
Eat's a lot of pizzas?
Exercises in the morning?
Yes?
No?
Yes?
No?
Unfit!
Fit
Fit
Unfit!

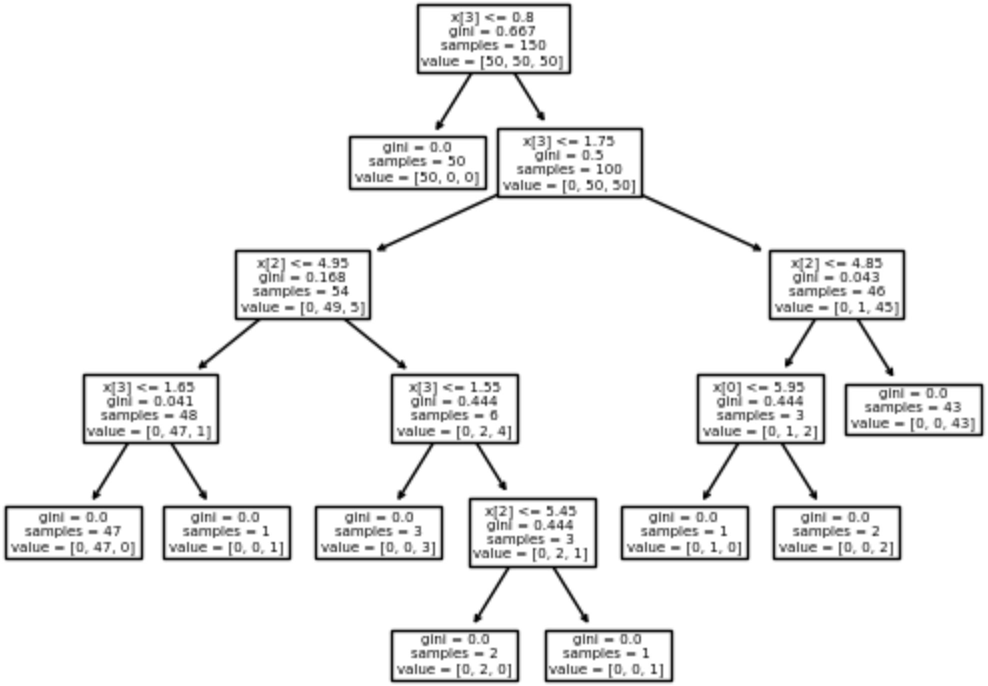
In [4]:

```python
# Source: https://scikit-learn.org/stable/modules/tree.html

from sklearn.datasets import load_iris
from sklearn import tree

iris = load_iris()
X, y = iris.data, iris.target

## Again, classifier is just two lines of code
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)

# You can even visualise the tree
tree.plot_tree(clf)
```

Out[4]:

```
[Text(0.5, 0.9166666666666666, 'x[3] <= 0.8\ngini = 0.667\nsamples =
150\nvalue = [50, 50, 50]'),
 Text(0.4230769230769231, 0.75, 'gini = 0.0\nsamples = 50\nvalue = [5
0, 0, 0]'),
 Text(0.5769230769230769, 0.75, 'x[3] <= 1.75\ngini = 0.5\nsamples =
100\nvalue = [0, 50, 50]'),
 Text(0.3076923076923077, 0.5833333333333334, 'x[2] <= 4.95\ngini =
0.168\nsamples = 54\nvalue = [0, 49, 5]'),
 Text(0.15384615384615385, 0.4166666666666667, 'x[3] <= 1.65\ngini =
0.041\nsamples = 48\nvalue = [0, 47, 1]'),
 Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples = 47\nvalue =
[0, 47, 0]'),
 Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue =
[0, 0, 1]'),
 Text(0.46153846153846156, 0.4166666666666667, 'x[3] <= 1.55\ngini =
```

```
0.444\nsamples = 6\nvalue = [0, 2, 4]'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 3\nvalue =
[0, 0, 3]'),
 Text(0.5384615384615384, 0.25, 'x[2] <= 5.45\ngini = 0.444\nsamples
= 3\nvalue = [0, 2, 1]'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.0\nsamples
= 2\nvalue = [0, 2, 0]'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsamples =
1\nvalue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.5833333333333334, 'x[2] <= 4.85\ngini =
0.043\nsamples = 46\nvalue = [0, 1, 45]'),
 Text(0.7692307692307693, 0.4166666666666667, 'x[0] <= 5.95\ngini =
0.444\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(0.6923076923076923, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0,
1, 0]'),
 Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0,
0, 2]'),
 Text(0.9230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples =
43\nvalue = [0, 0, 43]')]
```


x[3] <= 0.8
gini = 0.667
samples = 150
value = [50, 50, 50]
gini = 0.0
samples = 50
value = [50, 0, 0]
x[3] <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
x[2] <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
x[2] <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
x[3] <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
x[3] <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
x[0] <= 5.95
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 43
value = [0, 0, 43]
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
gini = 0.0
samples = 3
value = [0, 0, 3]
x[2] <= 5.45
gini = 0.444
samples = 3
value = [0, 2, 1]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]

Naive Bayes

Naive Bayes

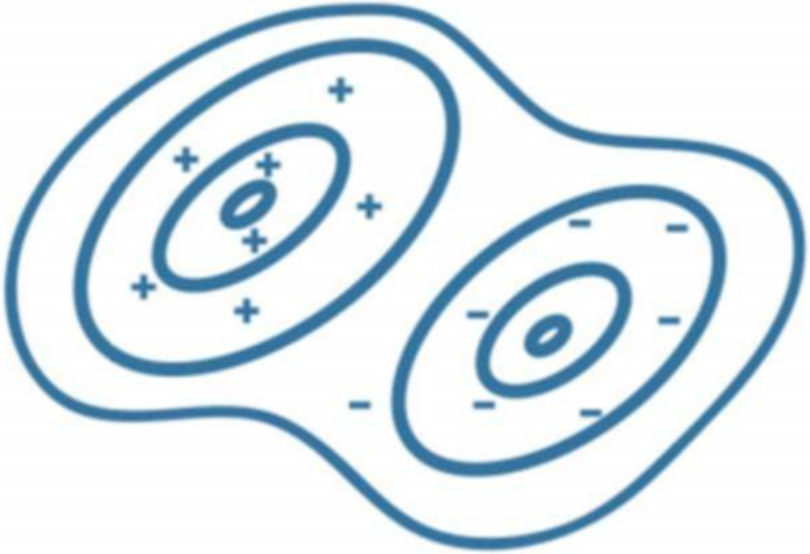
- Probabilistic classifier inspired by the Bayes theorem, assumes attributes are conditionally independent

Naive Bayes

- Probabilistic classifier inspired by the Bayes theorem, assumes attributes are conditionally independent
- Advantages: small amount of training data required, extremely fast

Naive Bayes

- Probabilistic classifier inspired by the Bayes theorem, assumes attributes are conditionally independent
- Advantages: small amount of training data required, extremely fast
- Disadvantages: zero probability problem, if the conditional probability is zero for a particular attribute...



Posterior

Attributes likelihoods

$$P(y|x_1,...,x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Predictor prior

Class prior

In [5]:

```
# Credit: https://www.kaggle.com/code/nizamudma/iris-data-classification-usin

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

#split data for train and test
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.15)

# No surpise, two lines
bc=GaussianNB()
bc.fit(x_train,y_train)

# Predict samples from the text set
print(bc.predict(x_test))
```

```
[0 1 2 1 1 0 0 2 0 2 0 0 0 2 1 1 1 0 0 2 2 0 0]
```

Support Vector Machine (SVM)

Support Vector Machine (SVM)

- Discriminative classifier defined by a separating hyperplane

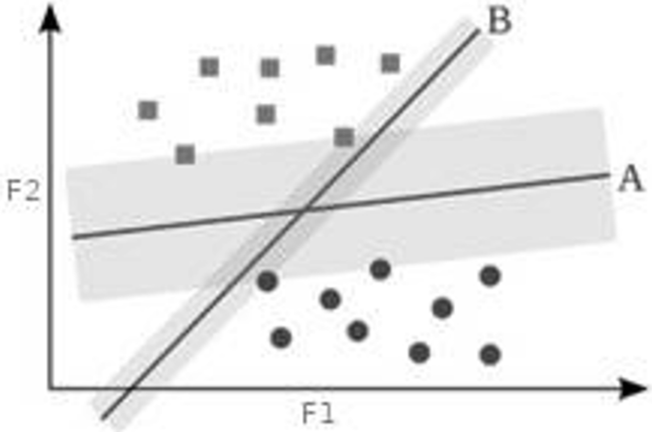
Support Vector Machine (SVM)

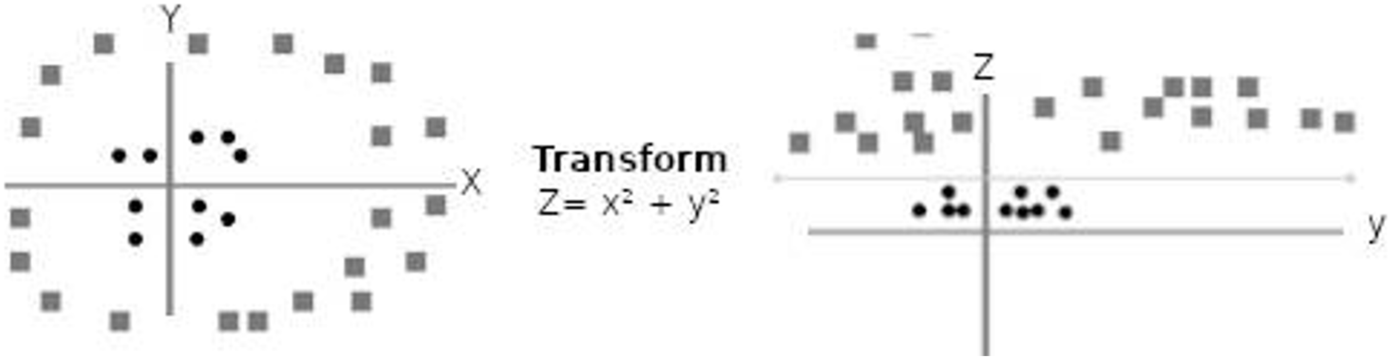
- Discriminative classifier defined by a separating hyperplane

- Tuning parameters in SVM classifier
    - Kernel - transformation method, e.g. Polynomial and exponential kernels
    - Regularisation - how much to avoid misclassifying each training example
    - Gamma - how far the influence of a single training example reaches, high gamma → only nearby examples

Support Vector Machine (SVM)

- Discriminative classifier defined by a separating hyperplane
- Tuning parameters in SVM classifier
  - Kernel - transformation method, e.g. Polynomial and exponential kernels
  - Regularisation - how much to avoid misclassifying each training example
  - Gamma - how far the influence of a single training example reaches, high gamma → only nearby examples
- A margin in SVM is a separation of line to the closest class points
  - A good margin is one where this separation is larger for both the classes


B
A
F2
F1

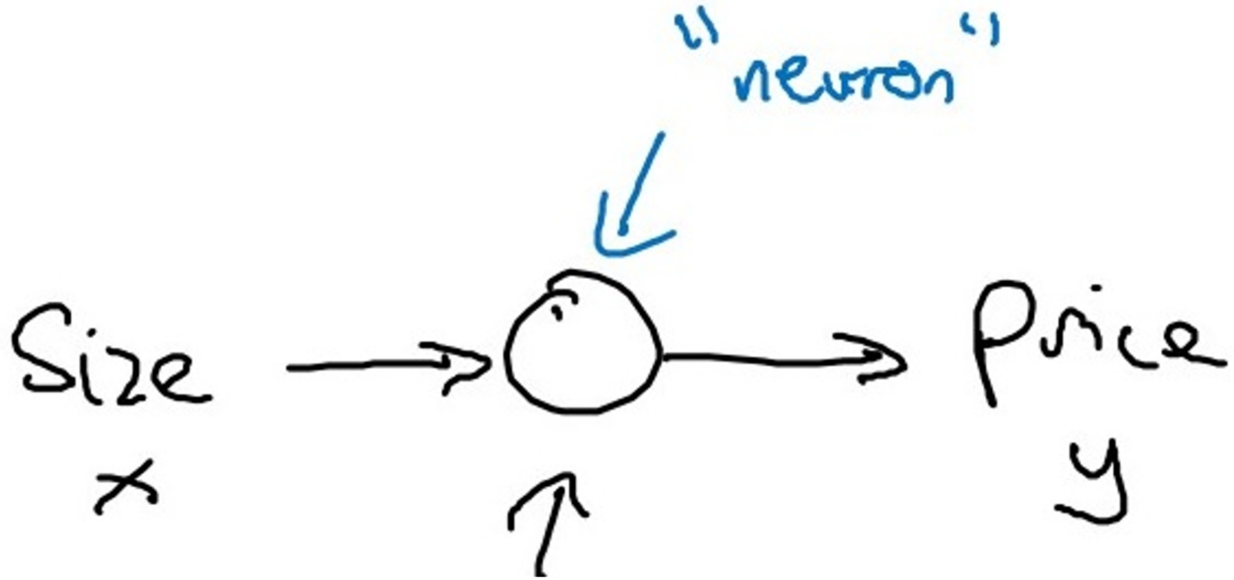

Y
X
Transform
$Z = x^2 + y^2$
Z
y

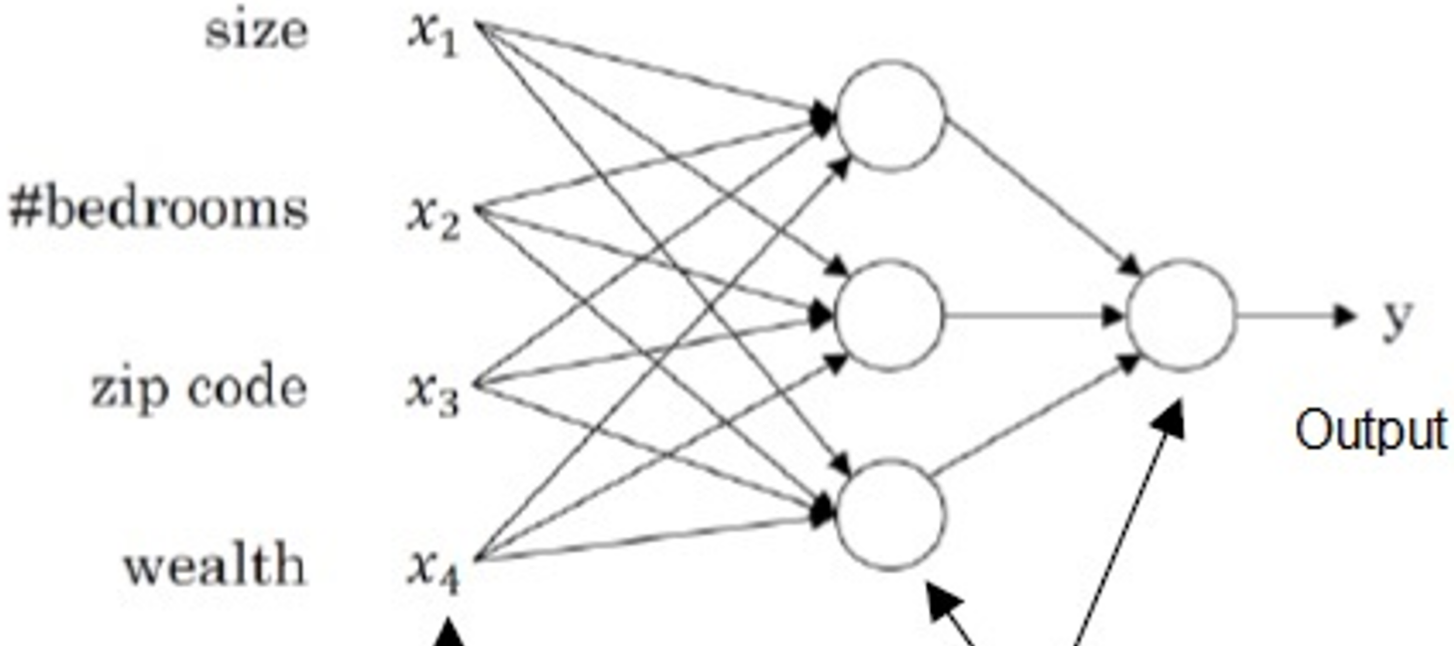
- We will do this one in Lab Topic 5! But trust me, again it is two lines of code...

Neural Network (NN)

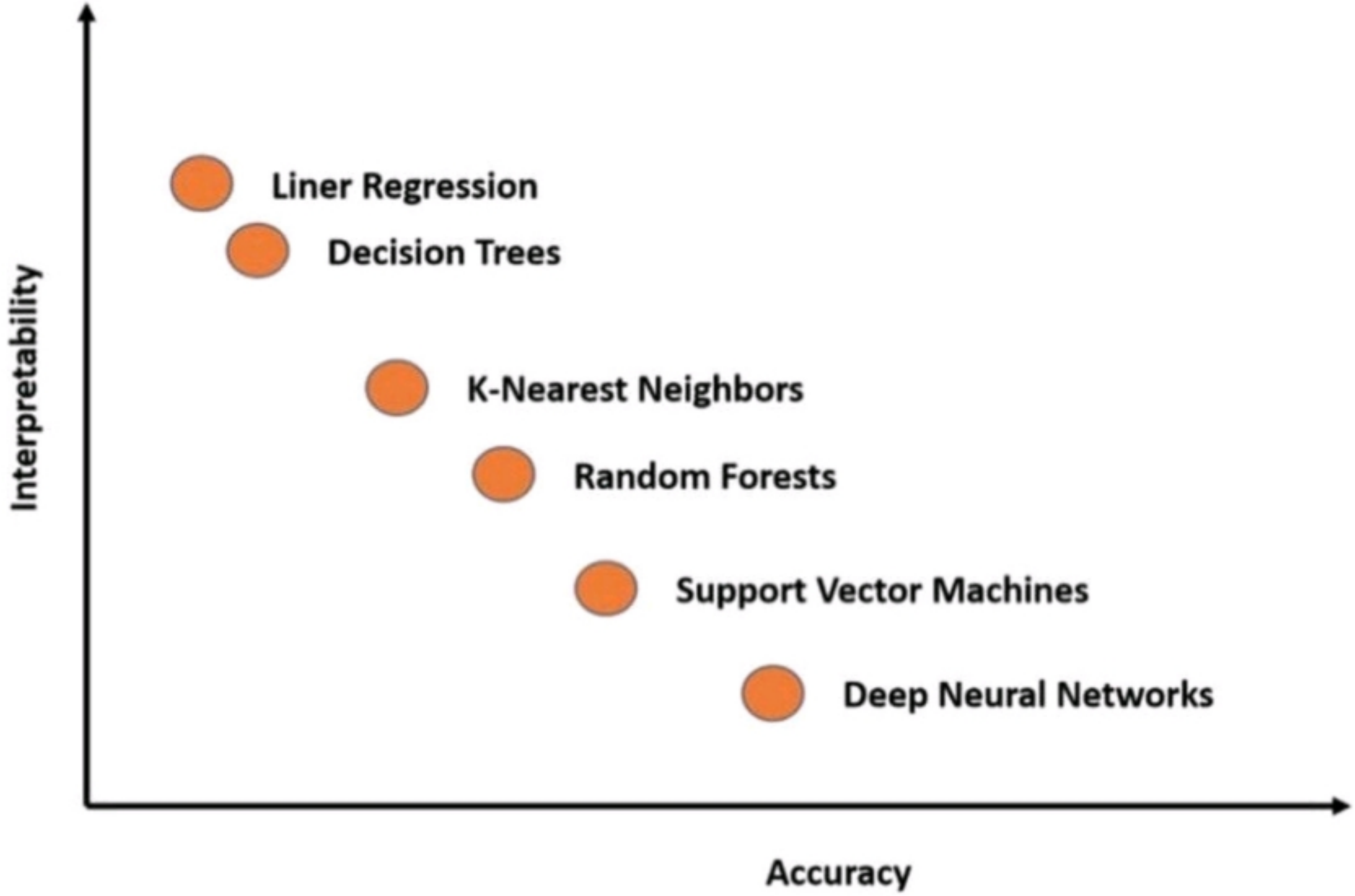
Neural Network (NN)

We will talk **EXTENSIVELY** about them later on


"neuron"
Size
x
Price
y


size
#bedrooms
zip code
wealth
y
Output

Accuracy and Interpretability



Semi-supervised Learning

Semi-supervised Learning

- Aims to label unlabelled data points using knowledge learned from a small number of labelled data points

Semi-supervised Learning

- Aims to label unlabelled data points using knowledge learned from a small number of labelled data points
- Learning with both unlabelled and labeled data points

Semi-supervised Learning

- Aims to label unlabelled data points using knowledge learned from a small number of labelled data points
- Learning with both unlabelled and labeled data points
- Used when large amounts of data are costly to label

Assumptions of Semi-supervised Learning

Assumptions of Semi-supervised Learning

- `Continuity` : Data points that are "close" have a common label

Assumptions of Semi-supervised Learning

- `Continuity` : Data points that are "close" have a common label
- `Cluster` : Data naturally forms discrete clusters, most common to share label

Assumptions of Semi-supervised Learning

- `Continuity` : Data points that are "close" have a common label
- `Cluster` : Data naturally forms discrete clusters, most common to share label
- `Manifold` : Data lies in a lower dimensional space than the input space

Examples of Semi-supervised Learning Algorithms

Examples of Semi-supervised Learning Algorithms

Transductive SVM

Examples of Semi-supervised Learning Algorithms

Transductive SVM

Label Propagation

Unsupervised Learning

Unsupervised Learning

- Does not have (or need) any labelled outputs, so its goal is to infer the natural structure present within a set of data points

Unsupervised Learning

- Does not have (or need) any labelled outputs, so its goal is to infer the natural structure present within a set of data points
- Finds inherent patterns of data

Unsupervised Learning

- Does not have (or need) any labelled outputs, so its goal is to infer the natural structure present within a set of data points
- Finds inherent patterns of data
- Most common tasks: **clustering** and **exploratory data analysis**

Examples of Unsupervised Learning Algorithms

Examples of Unsupervised Learning Algorithms

K-means Clustering

Examples of Unsupervised Learning Algorithms

K-means Clustering

Principal Component Analysis (PCA)

Examples of Unsupervised Learning Algorithms

K-means Clustering

Principal Component Analysis (PCA)

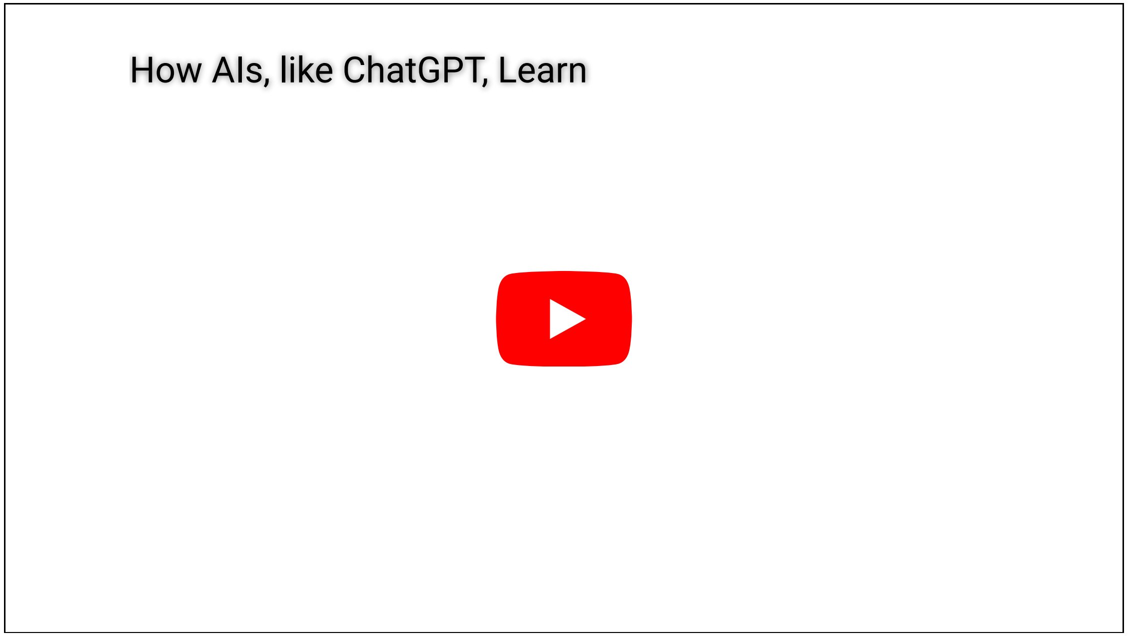
Autoencoders

A simpler explanation of what I just said

In [6]:

```python
import warnings
warnings.filterwarnings('ignore')
from IPython.display import HTML
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/R9OH
```

Out[6]:



As you can see, we can do more things beyond classification, such as **detecting** where the object lies in the image, **recognise** it from a pool of similar objects, **segment** where it is or even **track** it in a video feed!