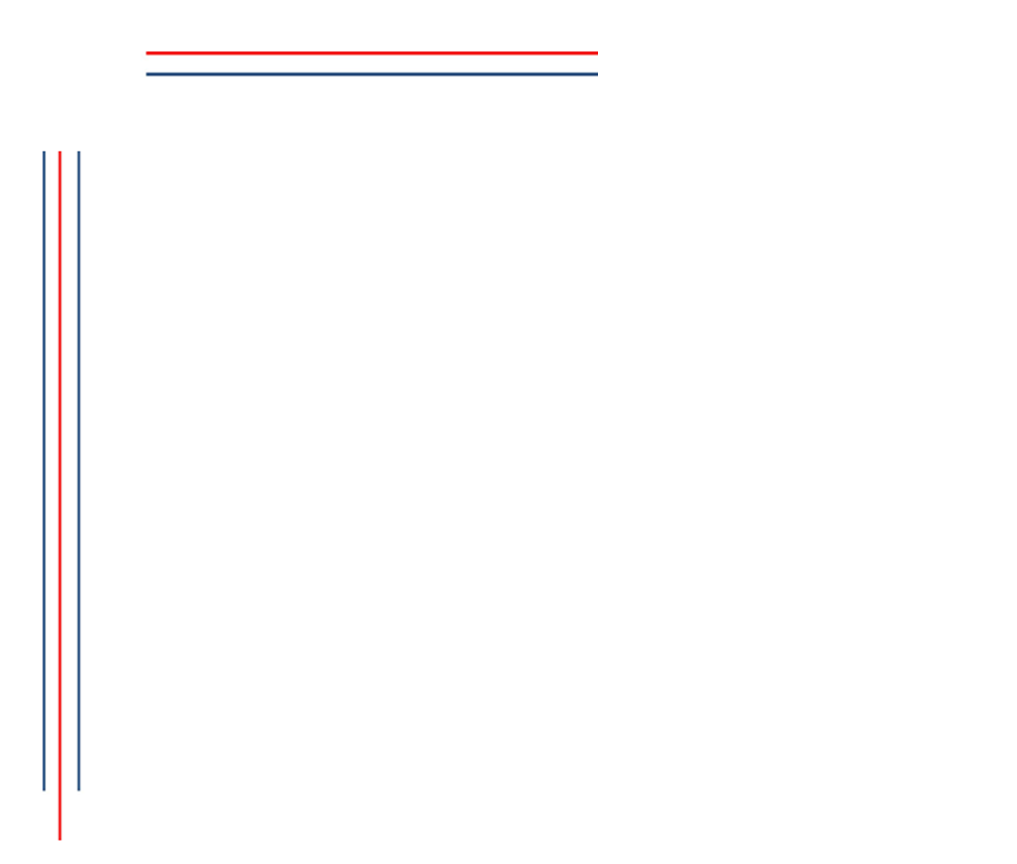
 **SEP SES TecNM**



**INSTITUTO TECNOLÓGICO DE TOLUCA**

**Inteligencia Artificial**

**UNIDAD 3**

Documentación de análisis de

Reconocimiento Facial

**C A R R E R A:**

Ingeniería en Sistemas Computacionales

**PRESENTA:**

Carrillo Ortiz Isaac Samuel 18280759

Eduardo Domínguez Cordero 18280735

Kevin Daniel Flores López 18280754

Nava Ceballos Emiliano Adolfo 18280750

**ASESOR:**

Angelo Marlon Montoya Guerra

**METEPEC, ESTADO DE MÉXICO, 25 DE MAYO DE 2021**

Índice

[Objetivo General 3](#_Toc72876304)

[Objetivos Específicos 3](#_Toc72876305)

[Planteamiento del problema 3](#_Toc72876306)

[Introducción 4](#_Toc72876307)

[Análisis 5](#_Toc72876308)

[Face Detection vs Face Recognition 5](#_Toc72876309)

[Face\_recognition 5](#_Toc72876310)

[Api.compare\_faces 6](#_Toc72876311)

[Api.face\_distance 6](#_Toc72876312)

[Api.face\_encodings 6](#_Toc72876313)

[Api.face\_landmarks 7](#_Toc72876314)

[Api.face\_locations 7](#_Toc72876315)

[Api.load\_image\_file 8](#_Toc72876316)

[Librería os 8](#_Toc72876317)

[os.getcwd() 8](#_Toc72876318)

[os.path.join(path, \*paths) 8](#_Toc72876319)

[Librería glob 9](#_Toc72876320)

[glob.glob 9](#_Toc72876321)

[Librería Numpy 9](#_Toc72876322)

[numpy.argmin 9](#_Toc72876323)

[Librería OpenCV 10](#_Toc72876324)

[Cv2.video\_capture () 10](#_Toc72876325)

[Cv2.resize 10](#_Toc72876326)

[Cv2.rectangle 10](#_Toc72876327)

[Cv2.putText 11](#_Toc72876328)

[Cv2.imshow() 11](#_Toc72876329)

[Cv2.waitkey 11](#_Toc72876330)

[Entrenando el programa 12](#_Toc72876331)

[Aprendizaje supervisado 12](#_Toc72876332)

[Algoritmo de Aprendizaje Supervisado: K-NN 12](#_Toc72876333)

[Imágenes como matrices 14](#_Toc72876334)

[Bibliografía 15](#_Toc72876335)

# Objetivo General

Analizar un proyecto de reconocimiento de patrones.

# Objetivos Específicos

Instalar un proyecto que utilice reconocimiento de patrones en una distribución de Linux.

Investigar sobre reconocimiento de patrones.

Investigar sobre las librerías y funciones que se utilizan en el programa.

# Planteamiento del problema

Detectar los rostros de diferentes imágenes por medio de una cámara web.

## 

# Introducción

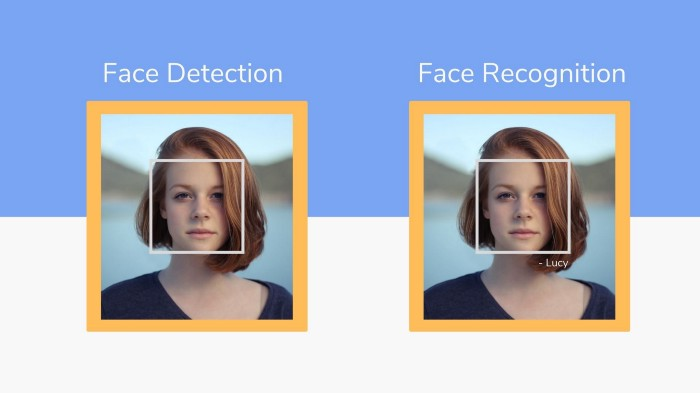
El reconocimiento de patrones (RP) es la disciplina científica cuya meta es la clasificación de objetos en cierto número de categorías o clases. Dependiendo del uso puede tratarse de imágenes, señales electromagnéticas o cualquier otro tipo de medida que requiera clasificación. El reconocimiento de patrones trata con problemas de clasificación que se desea sean delegados a una máquina, por ejemplo: visión de computadoras, escaneo de huellas, escaneo de iris, reconocimiento óptico de caracteres, etc.

El programa que se analizara a continuación permite reconocer el rostro de una persona por medio de una imagen utilizando una cámara web y desarrollado en python. Este programa hace uso de los algoritmos de aprendizaje supervisado.

# Análisis

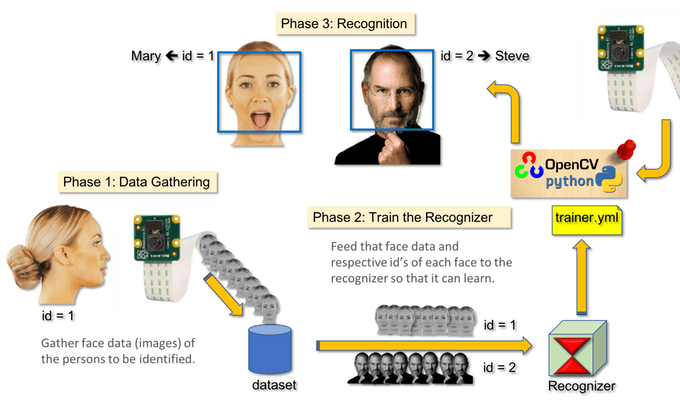
## Face Detection vs Face Recognition

Estas dos cosas pueden parecer muy similares, pero en realidad no son lo mismo. Entendamos la diferencia para no perder el punto. La detección de rostros es el proceso de detección de rostros, a partir de una imagen o un video que no importa. El programa no hace más que encontrar las caras. Pero, por otro lado, el reconocimiento facial, el programa que encuentra las caras y también puede decir qué cara pertenece a quién. Por lo tanto, es más informativo que solo detectarlos. Para escribir un código que reconozca caras necesita algunos datos de entrenamiento, debemos entrenar nuestra máquina para que conozca las caras y quiénes son. En este proyecto, somos nosotros los que enseñamos nuestro programa. En el aprendizaje automático, hay dos tipos de aprendizaje; supervisado y no supervisado. En este proyecto vamos a utilizar el aprendizaje supervisado.



## Face\_recognition

La biblioteca face\_recognition, creada por Adam Geitgey, incluye la funcionalidad de reconocimiento facial de dlib.



### Api.compare\_faces

**face\_recognition.api.compare\_faces(known\_face\_encodings, face\_encoding\_to\_check, tolerance=0.6)**

Compara una lista de codificaciones de caras con una codificación candidata para ver si coinciden.

Parámetros:

known\_face\_encodings: una lista de codificaciones de caras conocidas

face\_encoding\_to\_check: una codificación de una sola cara para comparar con la lista

tolerancia - Cuánta distancia entre caras para considerarlo una coincidencia. Más bajo es más estricto. 0,6 es el mejor rendimiento típico.

Devoluciones:

Una lista de valores verdaderos / falsos que indican qué codificaciones de caras conocidas coinciden con la codificación de caras para comprobar

### Api.face\_distance

**face\_recognition.api.face\_distance(face\_encodings, face\_to\_compare)**

Dada una lista de codificaciones de caras, compárelas con una codificación de caras conocida y obtenga una distancia euclidiana para cada cara de comparación. La distancia te dice qué tan similares son las caras.

Parámetros:

face\_encodings - Lista de codificaciones de caras para comparar

face\_to\_compare: una codificación de rostros con la que comparar

Devoluciones:

Un ndarray numpy con la distancia para cada cara en el mismo orden que la matriz "faces"

### Api.face\_encodings

**face\_recognition.api.face\_encodings(face\_image, known\_face\_locations=None, num\_jitters=1, model='small')**

Dada una imagen, devuelve la codificación de caras de 128 dimensiones para cada cara de la imagen.

Parámetros:

face\_image: la imagen que contiene una o más caras

known\_face\_locations - Opcional - los cuadros delimitadores de cada cara si ya los conoce.

num\_jitters: cuántas veces se debe volver a muestrear la cara al calcular la codificación. Más alto es más preciso, pero más lento (es decir, 100 es 100 veces más lento)

modelo - Opcional - qué modelo usar. “Grande” o “pequeño” (predeterminado) que solo devuelve 5 puntos pero es más rápido.

Devoluciones:

Una lista de codificaciones de caras de 128 dimensiones (una para cada cara de la imagen)

### Api.face\_landmarks

**face\_recognition.api.face\_landmarks(face\_image, face\_locations=None, model='large')**

Dada una imagen, devuelve un dictado de las ubicaciones de las características de la cara (ojos, nariz, etc.) para cada cara de la imagen.

Parámetros:

face\_image - imagen para buscar

face\_locations: opcionalmente, proporcione una lista de ubicaciones de rostros para verificar.

modelo - Opcional - qué modelo usar. “Grande” (predeterminado) o “pequeño” que solo devuelve 5 puntos pero es más rápido.

Devoluciones:

Una lista de dictados de ubicaciones de características faciales (ojos, nariz, etc.)

### Api.face\_locations

**face\_recognition.api.face\_locations(img, number\_of\_times\_to\_upsample=1, model='hog')**

Devuelve una matriz de cuadros delimitadores de rostros humanos en una imagen.

Parámetros:

img - Una imagen (como una matriz numpy)

number\_of\_times\_to\_upsample: cuántas veces se debe aumentar la imagen en busca de caras. Los números más altos encuentran caras más pequeñas.

modelo: qué modelo de detección de rostros usar. "Hog" es menos preciso pero más rápido en las CPU. “Cnn” es un modelo de aprendizaje profundo más preciso que está acelerado por GPU / CUDA (si está disponible). El valor predeterminado es "hog".

Devoluciones:

Una lista de tuplas de ubicaciones de caras encontradas en orden css (superior, derecha, inferior, izquierda)

### Api.load\_image\_file

**face\_recognition.api.load\_image\_file(file, mode='RGB')**

Carga un archivo de imagen (.jpg, .png, etc.) en una matriz numpy

Parámetros:

archivo: nombre de archivo de imagen u objeto de archivo para cargar

modo: formato al que convertir la imagen. Solo se admiten "RGB" (RGB de 8 bits, 3 canales) y "L" (blanco y negro).

Devoluciones:

contenido de la imagen como matriz numpy

## Librería os

### os.getcwd()

Parámetros

N / A

Valor devuelto

Este método devuelve el directorio de trabajo actual de un proceso.

### os.path.join(path, \*paths)

Parámetro:

ruta: un objeto similar a una ruta que representa una ruta del sistema de archivos.

\* ruta: un objeto similar a una ruta que representa una ruta del sistema de archivos. Representa los componentes de la ruta que se unirán.

Un objeto similar a una ruta es un objeto de cadena o bytes que representa una ruta.

Nota: La sintaxis especial \* args (aquí \* rutas) en las definiciones de funciones en Python se usa para pasar un número variable de argumentos a una función.

Tipo de retorno: este método devuelve una cadena que representa los componentes de la ruta concatenados.

## Librería glob

### glob.glob

**glob.glob**(*pathname*, *\**, *recursive=False*)

Devuelve una lista posiblemente vacía de nombres de ruta que coincidan con el nombre de ruta, que debe ser una cadena que contenga una especificación de ruta. El nombre de ruta puede ser absoluto (como /usr/src/Python-1.5/Makefile) o relativo (como ../../Tools/\*/\*.gif), y puede contener comodines de estilo shell. Los enlaces simbólicos rotos se incluyen en los resultados (como en el shell). La clasificación o no de los resultados depende del sistema de archivos. Si un archivo que cumple las condiciones se elimina o se agrega durante la llamada de esta función, no se especifica si se incluirá un nombre de ruta para ese archivo.

## Librería Numpy

### numpy.argmin

**numpy.argmin(*a*, *axis=None*, *out=None*)**

Devuelve los índices de los valores mínimos a lo largo de un eje.

Parámetros

aarray\_like

Matriz de entrada.

axisint, opcional

De forma predeterminada, el índice está en la matriz plana, de lo contrario, a lo largo del eje especificado.

outarray, opcional

Si se proporciona, el resultado se insertará en esta matriz. Debe tener la forma y el tipo adecuados.

Devoluciones

index\_arrayndarray de ints

Matriz de índices en la matriz. Tiene la misma forma que a.shape con la dimensión a lo largo del eje eliminado.

## Librería OpenCV

OpenCV (Open Source Computer Vision) es una librería open source de visión por computador, análisis de imagen y aprendizaje automático. Para ello dispone de infinitud de algoritmos que permiten, con sólo escribir unas pocas líneas de código, identificar rostros, reconocer objetos, clasificarlos, detectar movimientos de manos…

OpenCV es una librería multiplataforma disponible para Windows, Mac, Linux y Android distribuida bajo licencia BSD. Puede programarse con C, C++, Python, Java y Matlab.

### Cv2.video\_capture ()

Este es un método el cual nos permitirá iniciar el proceso de captura del video. Por ejemplo: cv2.video\_capture(0).

El número dentro de los paréntesis indica que cámara se usa para la captura, si se tiene una cámara conectada se puede colocar 0 o -1. En el caso de que hubiese más cámaras conectadas y se deseara trabajar alguna de ellas se tendría que especificar con cual, 1, 2 o 3, por ejemplo.

### Cv2.resize

Este método nos sirve para cambiar el tamaño de una imagen en Python, puede usar la función cv2.resize () de la biblioteca OpenCV cv2. El cambio de tamaño, de forma predeterminada, solo cambia el ancho y el alto de la imagen. La relación de aspecto se puede conservar o no, según el requisito.

sintaxis de la función cv2.resize ()

cv2.resize (src, dsize, fx , fy )

dónde

src es la imagen de origen, original o de entrada en forma de matriz numpy

dsize es el tamaño deseado de la imagen de salida, dado como tupla

fx es el factor de escala a lo largo del eje X o el eje horizontal

fy es el factor de escala a lo largo del eje Y o el eje vertical

.

### Cv2.rectangle

El método es usado para dibujar un rectángulo en cualquier imagen, las coordenadas son representadas como tuplas de dos valores.

### Cv2.putText

Este método dibuja la cadena de texto especificada.

Sintaxis:   
cv.putText(img, texto, esq, fuente, tam, color[, grosor[, tipoLinea[, origenInferiorIzuierdo]]) ->img

dónde

Img es la imagen.

texto, es la cadena de texto a dibujar.

esq es la esquina inferior izquierda del texto, en la imagen

fuente es el tipo de fuente

tam es el tamaño de la fuente

color es el color del texto.

grosor es el grosor del texto

tipoLinea es el tipo de línea

origenInferiorIzquierdo], si es verdadero, el origen de la imagen está en la esquina inferior izquierda, de lo contrario, el origen está en la esquina superior izquierda.

### Cv2.imshow()

El método cv2.imshow () se usa para mostrar una imagen en una ventana. La ventana se ajusta automáticamente al tamaño de la imagen.

Sintaxis: cv2.imshow(window\_name, imagen)

Parámetros: window\_name: una cadena que representa el nombre de la ventana en la que se mostrará la imagen.

Imagen: Es la imagen que se va a mostrar.

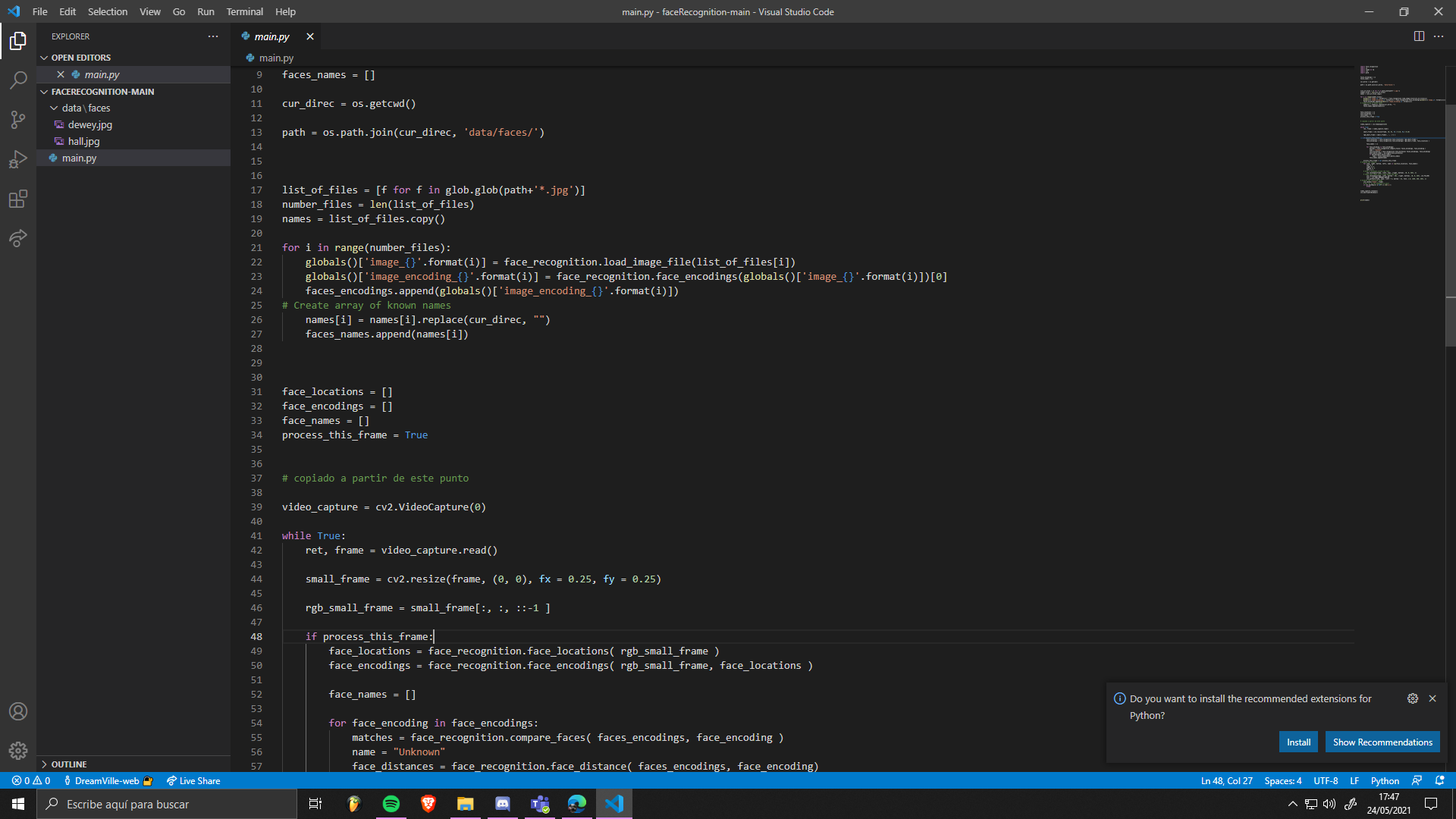
Valor de retorno: no devuelve nada.

### Cv2.waitkey

Cv2.waitKey(1) mostrará un cuadro durante 1 ms, después de lo cual la pantalla se cerrará automáticamente. Dado que el sistema operativo tiene un tiempo mínimo entre los subprocesos de conmutación, la función no esperará exactamente 1 ms, esperará al menos 1 ms, dependiendo de qué más se esté ejecutando en su computadora en ese momento.

## Entrenando el programa

En esta parte del código, el programa reconoce/aprende a quien pertenece la imagen(cara) que se está visualizado en ese momento, Esto es posible renombrando esas imágenes. En este caso se utiliza el método de aprendizaje supervisado



## Aprendizaje supervisado

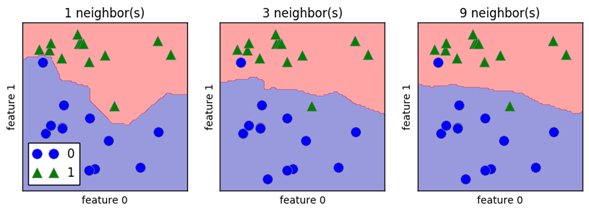
Los modelos de aprendizaje supervisado son aquellos en los que se aprenden funciones, relaciones que asocian entradas con salidas, por lo que se ajustan a un conjunto de ejemplos de los que conocemos la relación entre la entrada y la salida deseada. Este hecho incluso llega a proporcionar una de las clasificaciones más habituales en el tipo de algoritmos que se desarrollan, así, dependiendo del tipo de salida, suele darse una subcategoría que diferencia entre modelos de clasificación, si la salida es un valor categórico (por ejemplo, una enumeración, o un conjunto finito de clases), y modelos de regresión, si la salida es un valor de un espacio continuo.

### Algoritmo de Aprendizaje Supervisado: K-NN

El algoritmo de los k vecinos más cercanos (k-NN, o k Nearest Neighbour) es un algoritmo de clasificación supervisado basado en criterios de vecindad. En particular, k-NN se basa en la idea de que los nuevos ejemplos serán clasificados con la misma clase que tengan la mayor cantidad de vecinos más parecidos a ellos del conjunto de entrenamiento. Así pues, este algoritmo sigue un procedimiento que seguimos cada uno de nosotros al ver un ejemplo nuevo: vemos a qué se parece más de lo que conocemos, y lo metemos en la misma bolsa. Obviamente, este algoritmo introduce ya una condición que debe cumplirse entre los datos que tengamos, y es que hemos de ser capaces de medir la similaridad entre dos cualesquiera de ellos, por eso se considera que el espacio de datos de entrada debe ser algo parecido a un espacio métrico (es decir, un espacio donde haya una distancia definida), por lo que muchas veces será común pensar en los datos de entrada como si vinieran dados por medio de vectores de un espacio vectorial numérico estándar.

Su versión más simple, el algoritmo del vecino más cercano (aquel que asigna a una nueva muestra la clasificación de la muestra más parecida) explora todo el conocimiento almacenado en el conjunto de entrenamiento para determinar cuál será la clase a la que pertenece una nueva muestra, pero únicamente tiene en cuenta el vecino más próximo (más similar) a ella, por lo que es lógico pensar que es posible que no se esté aprovechando de forma eficiente toda la información que se podría extraer del conjunto de entrenamiento.

Con el objetivo de resolver esta posible deficiencia surge la generalización de los k vecinos más cercanos (k-NN), en la que se utiliza la información suministrada por los k ejemplos del conjunto de entrenamiento más cercanos al que queremos clasificar.



En el algoritmo k-NN existe el problema de que requiere de mucha memoria y tiempo de ejecución porque hay que almacenar continuamente todos los datos que definen el espacio de ejemplos inicial. Sin embargo, es muy probable que muchas de las muestras iniciales no sean necesarias para clasificar las demás, ya que su información es redundante con las otras existentes. Algunas variantes interesantes que intentan mitigar este problema son:

1. k-NN Condensado: Dado un orden en los datos de entrada, cada ejemplo del conjunto se clasifica por medio de k-NN haciendo uso únicamente de los datos anteriores; si la clasificación obtenida coincide con la real, ese ejemplo se elimina de los datos, si no, permanece. Observa que depende del orden dado a los datos y, además, tiene el problema de conservar los datos que introducen ruido al sistema.
2. k-NN Reducido: es similar a la anterior, pero se comienza con el conjunto completo de datos, y se eliminan aquellos que no afectan a la clasificación del resto de datos de entrada. Al revés de lo que ocurre con la condensación, este método es capaz de eliminar las muestras que producen ruido, y guarda aquellas que son críticas para la clasificación.

## Imágenes como matrices

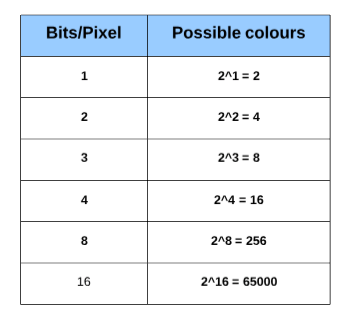
Una imagen no es más que una matriz estándar de Numpy que contiene píxeles de puntos de datos. Cuanto mayor sea el número de píxeles en una imagen, mejor es su resolución.

Puede pensar que los píxeles son pequeños bloques de información dispuestos en forma de una cuadrícula 2D, y la profundidad de un píxel se refiere a la información de color presente en ella.

Para ser procesado por una computadora, una imagen debe convertirse en una forma binaria. El color de una imagen se puede calcular de la siguiente manera:

*Número de colores / sombras = 2^bpp (donde bpp representa bits por pixel)*

Por lo tanto, cuanta más cantidad de bits/píxel, más colores posibles en las imágenes. La siguiente tabla muestra dicha relación de forma más clara:



# Bibliografía

Behic Guven. (2020). Building a Face Recognizer in Python. 25/05/2021, de towardsdatascience.com Sitio web: <https://towardsdatascience.com/building-a-face-recognizer-in-python-7fd6630c6340>

OpenCV . OpenCV-Python Tutorials. 25/05/2021, de OpenCV Sitio web: <https://docs.opencv.org/master/d6/d00/tutorial_py_root.html>