UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica

Deterioro cognitivo leve (DCL): ¿Interacción o aislamiento?

Una herramienta basada en memoria digital para el entramiento cognitivo de pacientes afectos por DCL.

Autor: Rubén Nieto Vargas

Directora:	**Dra. Mariella Dimiccoli**
Realizado en:	**Departament de Matemàtiques i Informàtica**
Barcelona,	**26 de enero de 2017**

Abstract

This project, in the context of "La Marató de TV3", pretends to develop a tool based on digital memories for the cognitive training of elderly people affected by mild cognitive impairment. Unfortunately, these people are highly exposed to the risk of suffering from the Alzheimer disease. The aim of the cognitive training based on digital living memories is to delay the deterioration of the memory and the cognitive functions in general.

The images of social interactions, due to the emotional charge that they carry on, trigger a strong response of autobiographical memory and, consequently, are very suitable to stimulate it.

For this reason, this work will develop an algorithm that, given a group of images captured by a patient with a portable camera during a long period of time (no shorter than two weeks), will determine with how many people the subject usually interacts and with who interacts the most. For each person, a group of images will be selected to be used so that the patient will be able to recognise his relatives in different contexts and so that he could go back to revive the moment and consolidate his memory.

For each person who the patient use to interact, it takes several images that are being used for the patients could recognize them in different situations, could revived the moment and consolidate their thoughs.

This dissertation has helped me to deepen in two aspects which are vital for the visionIi have of the world. In the first place, being able to help the elderly people, who are the reason why we are who we are. Secondly, computer vision and machine learning which are my passion.

Resum

En aquest projecte, dins del marc de la Marató de TV3, es pretén desenvolupar una eina basada en memòria digital per l'entrenament cognitiu de la gent gran afectada per la deterioració cognitiva lleu. Per desgràcia, aquestes persones estan exposades al risc de patir Alzheimer. L'objectiu de l'entrenament cognitiu basat en la memòria digital és poder retardar la deterioració de la memòria i de les seves funcions cognitives en general.

Les imatges de les interaccions socials, a causa de la càrrega emocional que tenen, desencadenen en una forta resposta de la memòria autobiogràfica i per tant, són adequades per a estimular.

Per tant, en aquest treball, desenvoluparem un algorisme que, donat un conjunt d'imatges capturades per un pacient amb una càmera portàtil durant un llarg període de temps, recomanable un mínim de dues setmanes, es determina amb quantes persones interactua normalment i amb quines interactua més. Per a cada persona, se seleccionarà un conjunt d'imatges que s'utilitzaran amb la intenció que el pacient sigui capaç de reconèixer als seus familiars en diferents contextos i pugui tornar a viure el moment i consolidar la seva memòria.

Per a cada persona amb la qual el pacient normalment interactua, es seleccionarà un conjunt d'imatges que s'utilitzaran perquè el pacient sigui capaç de reconèixer en diferents contexts, pugui tornar a viure el moment i consolidar la seva memòria.

Aquest treball m'ha servit per aprofundir en dos aspectes que són vitals per a la meva visió del món. En primer lloc, poder col·laborar i ajudar a la gent gran, que són els responsables directes de ser els qui som, l'altre aspecte, és la visió per computador i l'aprenentatge automàtic.

Resumen

En este proyecto, dentro del marco de “la Marató de TV3”, se pretende desarrollar una herramienta basada en memorias digitales para el entrenamiento cognitivo de la gente mayor afectada por deterioro cognitivo leve. Por desgracia, estas personas están muy expuestas al riesgo de padecer Alzheimer. El objetivo del entrenamiento cognitivo basado en memoria digital es de retrasar el deterioro de la memoria y de las funciones cognitivas en general.

La imágenes de interacciones sociales, debido a la carga emocional que conllevan, desencadenan una fuerte respuesta de la memoria autobiográfica y por lo tanto son muy adecuadas para estimularla.

Por lo tanto, en este trabajo desarrollamos un algoritmo que, dado un conjunto de imágenes capturadas por un paciente a través de una cámara portátil durante un largo periodo de tiempo (mínimo dos semanas), se determina con cuántas personas suele interactuar y con quienes interactúa más. Para cada persona se selecciona un conjunto de imágenes que se utilizarán para que el paciente sea capaz de reconocer sus familiares en diferentes contextos y pueda revivir el momento y consolidar su memoria.

Para cada persona con la cual el paciente suele interactuar, se selecciona un conjunto de imágenes que se utilizarán para que el paciente sea capaz de reconocerla en diferentes contextos, pueda revivir el momento y consolidar su memoria.

Este trabajo me ha servido para profundizar en dos aspectos que son vitales para mi visión del mundo. En primer lugar, poder colaborar i ayudar a la gente mayor, que son los responsables directos de ser quienes somos, el otro aspecto, es la visión por computador i el aprendizaje automático.

Agradecimientos

Quiero agradecer a todos los docentes, que han dedicado parte de su vida en enseñar al resto sus conocimientos. A mi familia por prestarme soporte en todo momento. Y por último, pero no por ello menos importante, a mis amigos por darme la distracción necesaria i poder disfrutar de ellos en los mejores momentos. **Gracias a todos.**

Índice

1. Introducción **3**

1.1. Motivo y objetivo del trabajo 3

1.2. Estado del arte 3

1.3. Enfoque propuesto 3

1.4. Contribuciones 4

2. Metodología **5**

2.1. Creación del *dataset* 5

2.2. Detección facial 6

2.2.1. OpenFace 6

2.2.2. Face detection pose estimation and landmark localization in the wild 8

2.3. Cálculo de la similitud entre caras 9

2.3.1. Deep - Matching 9

2.3.2. Matriz de distancia 10

2.4. Métodos automáticos para el clustering de caras 11

2.4.1. Aprendizaje supervisado 11

2.4.2. Aprendizaje no supervisado 12

2.5. Visualización de los resultados obtenidos 17

2.5.1. Reducción de la dimensionalidad 17

2.5.2. PCA 17

2.5.3. t-SNE 18

3. Pruebas y resultados **19**

3.1. Creación del *dataset* 19

3.2. Detección facial 20

3.3. Evaluación de la detección facial 20

3.4. Búsqueda de la similitud entre caras 24

3.4.1. Matriz de distancias 24

3.4.2. Deep - Matching 25

3.5. Agrupaciones por personas 26

3.6. Visualización 29

3.6.1. Reducción de dimensionalidad 30

4. Estructura del código **32**

5. Conclusiones y líneas futuras **34**

A. Instalación del código **36**

B. Coste aproximado del proyecto **37**

C. Envío de correo electrónico **38**

D. Web de presentación **40**

E. Manual de usuario **43**

F. Bag of Tracklets **45**

Índice de figuras

1. Ejemplo de la estructura necesaria para la correcta ejecución. 6
2. Detección de cara con OpenFace 7
3. Metodología OpenFace 8
4. Metodología OpenFace 9
5. Funcionamiento Deep - Matching 10
6. Regresión lineal 12
7. Funcionamiento K - Means 13
8. Hierarchical Clustering en modo Agglomerative 14
9. Mean Shift, funcionamiento 15
10. DBSCAN 16
11. Calculo del área del rectángulo interior 21
12. Cálculo de precisión y *recall* 23
13. Caras diferentes. 24
14. Caras diferentes. 25
15. Resultados de los diferentes métodos de clustering 27
16. Estructura necesaria para la clasificación 28
17. Inicio de la interfície gráfica 29
18. Reducción de los datos utilizando t-SNE 30
19. Reducción de los datos utilizando PCA 31
20. Resultado de la ejecución correcta 38
21. Error durante la ejecución 39
22. Breve explicación sobre los objetivos 40
23. Breve explicación la metodología 41
24. Algoritmos utilizados 42
25. Menú de la aplicación 43
26. Menú para la reducción de dimensionalidad 43
27. Menú para la evaluación de los métodos utilizados 44
28. Bag of tracklet, ejemplo de la estructura obtenida 45
29. Clustering con los resultados de Bag of tracklet 46

Índice de cuadros

1. Tabla comparativa con los dos algoritmos utilizados 23
2. Ejemplo matriz de distancia 25

1. Introducción

1.1. Motivo y objetivo del trabajo

En este proyecto, desarrollaremos un algoritmo para detectar automáticamente a las personas con las cuales el usuario interactúa más. Esto se efectuará a través de una cámara que portarán, en este caso, las personas con deterioro cognitivo leve.

Nos centraremos en encontrar con quien interactúa el portador de la cámara, nos podremos preguntar con quien interactúa más, y así, ayudar a los psicólogos a preparar los ejercicios de memoria con las personas encontradas en las imágenes grabadas. Y, por último, analizaremos las interacciones encontradas buscando las horas que más interactúa con determinadas personas.

Por tanto, hablaremos de tres aspectos básicos para mi motivación personal. El primero, será el reconocimiento facial mediante dos algoritmos que explicaremos más adelante, para encontrar a las personas que interactúan con las personas portadoras de las cámaras que en este caso serán personas con deterioro cognitivo leve. Son imágenes de una cámara en movimiento y por lo tanto, puede que el paciente interactúe con alguien y que la cámara no lo capte bien, por eso es importante tener la secuencia completa y poder analizarlo correctamente. El segundo aspecto que trataremos será la comparación entre las diferentes personas que aparecen en las secuencias para poder determinar si se tratan de la misma persona. Y por último, tendremos que clasificar mediante métodos de aprendizaje automático para poder encontrar y cuantificar a las personas y ver las interacciones.

1.2. Estado del arte

En la visión por computador aún queda mucho por descubrir, en particular, la detección de caras, aunque se ha avanzado mucho en estos último años, aún queda mucho trabajo por hacer. Existen algoritmos muy buenos capaces de detectar caras, como el que veremos en este proyecto pero todavía se tiene que avanzar hacia el reconocimiento y agrupamiento de las personas. Es decir, saber quienes son las personas que aparecen en cada foto. En este proyecto, profundizaremos en sobre este tema y veremos los diferentes métodos existentes para ello.

Quizás, los puntos que nos encontraremos con más dificultades para desarrollar serán en los aspectos más innovadores puesto que hay poca información. Pero, esa es precisamente la motivación de este trabajo, descubrir y buscar nuevos algoritmos.

1.3. Enfoque propuesto

Como hemos visto en los objetivos, buscamos la interacción que tiene el usuario portador de la cámara. Para ello, en primer lugar buscaremos algoritmos de detección facial y veremos cuál se adecúa más a nuestras necesidades.

En segundo lugar, analizaremos la detección para poder obtener una compa-

ración entre las caras encontradas y así poder clasificar a las diferentes personas encontradas.

En tercer lugar, necesitaremos tratar nuestro *dataset* y prepararlo para poder medir los algoritmos utilizados, para la detección facial y así comparar los métodos utilizados y también para el aprendizaje automático.

Y por último, visualizaremos los resultados obtenidos mediante una interfície gráfica, en la cual mostraremos las interacciones de cada personas en las horas que se han llevado a cabo dichas interacciones.

1.4. Contribuciones

Las contribuciones de este proyecto han sido las siguientes:

- En primer lugar, se ha llevado a cabo la creación de un *dataset* con el que trabajar. El *dataset* consiste de N imágenes capturadas por una misma persona durante aproximadamente unas 4 semanas. Se han anotado las Bounding Boxes de las caras y se ha creado el Ground Truth de los clústers de cara.
- Se ha buscado cual es el mejor algoritmo para la detección facial en nuestro caso, los que hemos utilizado han sido:
  - El primero es **OpenFace**, un algoritmo desarrollado en Python y Open-Source, fue creado en 2015. [2] [25]
  - Y, **FaceDetection**, un algoritmo para Matlab desarrollado en 2013. [1]
- El cálculo de la similitud de las caras se ha desarrollado con la ayuda de dos componentes.
  - El primero, se ha calculado mediante la obtención de las características de OpenFace y algoritmos de cálculo de distancias.
  - Y en segundo lugar, se ha utilizado un algoritmo llamado **Deep-Matching**. [9]
- Se han utilizado métodos de aprendizaje supervisado y algoritmos de aprendizaje no supervisado. Con el fin de agrupar las caras de una misma persona y obtener a todas las personas que tenemos en nuestro *dataset*. [3][16] [15] [17]
- Se ha desarrolado una interfície gráfica para la visualización: para ello se ha utilizado *Tkinter*; Además, se pueden visualizar las características de las caras mediante algoritmos de reducción de dimensionalidad como *PCA y t-SNE*. [4]

2. Metodología

En esta sección, explicaremos todos los pasos necesarios para completar este proyecto y comparar entre los diferentes métodos que tenemos. Separaremos esta sección en los cinco pilares fundamentales de esta búsqueda.

- En primer lugar, prepararemos el *dataset* con el que trabajaremos
- En segundo lugar, tendremos la detección facial
- En el siguiente paso, veremos como obtener la matriz de distancia o bien, la matriz de similitud, dependiendo del algoritmo que utilicemos.
- El próximo paso, será poder hacer una reducción de dimensionalidad para poder visualizarlo mejor
- También, tendremos que agrupar a cada persona mediante técnicas de *clustering*.
- Y por último, visualizaremos todas las imágenes encontradas para una persona.

2.1. Creación del *dataset*

En esta sección explicaremos los pasos necesarios para la preparación de las imágenes que queremos analizar y encontrar las interacciones.

En primer lugar, prepararemos la estructura de carpetas que debe tener nuestro *dataset* para poder utilizarlo en nuestra aplicación. Para ello, debemos tener las secuencias, es decir, los diferentes momentos grabados, separados en carpetas y dentro de estos deberá existir una carpeta que se llame *data*. Esto es muy importante, puesto que es la forma que tenemos organizado nuestro código.

Como podemos observar en la imagen 1, será la estructura que necesitaremos tener, con indiferencia de los nombres de las imágenes, pero, si que deberán estar en formato *JPG*.

En segundo lugar, necesitaremos crear el *Ground Truth*, es decir, obtener todas las caras de nuestro *dataset* de forma manual y totalmente supervisada, es decir, sin utilizar ningún tipo de algoritmo para la detección. Este paso es muy importante, pues, será el paso necesario para poder evaluar los diferentes métodos utilizados para la detección facial. Para ello, tendremos todas las personas que aparecen en las diferentes secuencias con sus *Bounding Box*, es decir, con la "caja" que nos indica en que puntos de la imagen se encuentran las diferentes caras.

Y por último, será necesario preparar el *dataset* para evaluar los diferentes métodos de aprendizaje automático utilizados. Para ello, tendremos que buscar y agrupar a todas las personas que aparezcan en nuestro *dataset* y así, poder obtener su *Ground Truth* para más tarde comparar las agrupaciones efectuadas por los métodos utilizados y ver cuál se adecúa mejor a nuestro propósito.

Estructura necesaria

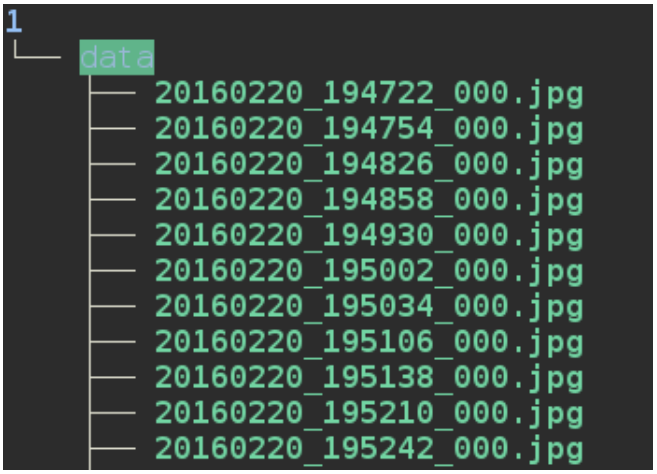


Figura 1: Ejemplo de la estructura necesaria para la correcta ejecución.

2.2. Detección facial

El primer paso será obtener las imágenes que aparecen caras. Para ello, utilizaremos dos algoritmos diferentes para dicha detección. Con ello, podremos observar que algoritmo nos resulta más útil. El primer algoritmo que utilizaremos será OpenFace, para *Python*. El segundo algoritmo, para *Matlab*, llamado *Face Detection pose estimation and landmark localization in the wild*.

En este paso tenemos como objetivo obtener las caras en las diferentes imágenes, es el primer paso y el fundamental para poder desarrollar este proyecto. Antes de utilizar un método u otro, es fundamental entender que necesitamos. Nosotros, en concreto, necesitamos obtener la matriz de características de cada cara que interactúe con el paciente. Para ello, tenemos que tener presente que no querremos las caras distantes y pequeñas, pues eso, es un indicador de que no hay interacción y por tanto, las rechazaremos. Un segundo requisito es que la *Bounding Box*, es decir, la caja contenedora de la cara, debe tener un formato un poco más grande que la simple cara, pues puede ser objeto de investigación el escenario donde se encuentra la persona que detectamos.

2.2.1. OpenFace

OpenFace es un algoritmo muy potente de código abierto, con él, podremos pasarle un conjunto de imágenes y obtendremos las imágenes donde aparecen caras, compararlas y así obtener una distancia. Más tarde, con esta distancia podremos utilizar métodos de *Machine Learning*, que explicaremos más adelante.

Este algoritmo tiene una función de ejemplo que se puede utilizar en el navegador y así explorar todos los casos que detecta la cara y así aprender y probar antes de utilizar el código. Como podemos observar en la figura 2, es el entorno creado para probar el algoritmo de *OpenFace*.

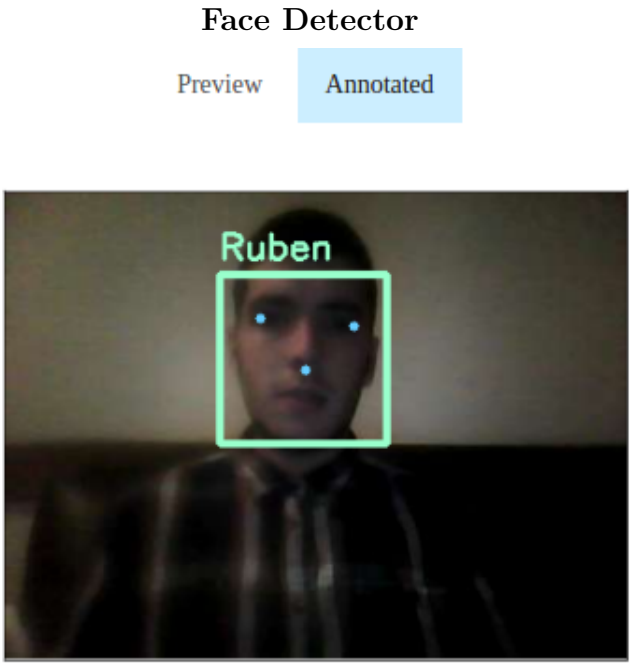


Figura 2: Detección de cara con OpenFace

Como hemos mencionado con anterioridad, buscamos obtener las características de las caras, esto es, una matriz con los datos necesarios de cada cara. Necesitamos dichas características ya que nuestro propósito será comparar las caras, este proceso de comparación lo explicaremos más adelante. *OpenFace* nos dará esta matriz para cada cara con dichas características que buscamos con una dimensión de 128, y obviamente, también nos dará la información necesaria que nos dirá en que puntos de la imagen se encuentra la cara.

Con el fin de tener el código más encapsulado, primero, recorreremos todas las imágenes para obtener las imágenes que aparecen caras, a continuación, obtendremos la matriz de características de cada cara encontrada. Con tal de agilizar este paso, haremos que el código sea multi-threading. Para ello, crearemos un hilo (thread) por cada imagen y así obtener la matriz de características de cada cara de forma más rápida, guardaremos cada matriz un fichero de formato *txt*. Así como también la *Bounding Box* de la cara en un fichero *mat* (para *Matlab*) que contendrá el punto más a la izquierda de la caja contenedora y más a la derecha del eje de las equis y, el punto más alto y más bajo del eje de las íes.

Como vemos en la figura siguiente (figura 3), extraída de la web oficial de *OpenFace*, los pasos a a seguir son los siguientes, primero leemos la imagen que queremos detectar las caras, para más tarde hacer la detección y acto seguido se hace la transformación mediante redes neuronales, este proceso es invisible para nosotros, es decir, no hará falta que modifiquemos el código para la detección, ni saber el funcionamiento de la red neuronal. Nos podemos ceñir a introducir la imagen y

obtener los resultados, pero, en nuestro caso, si que añadiremos funcionalidad al código, con el fin de adaptarlo a nuestras necesidades.

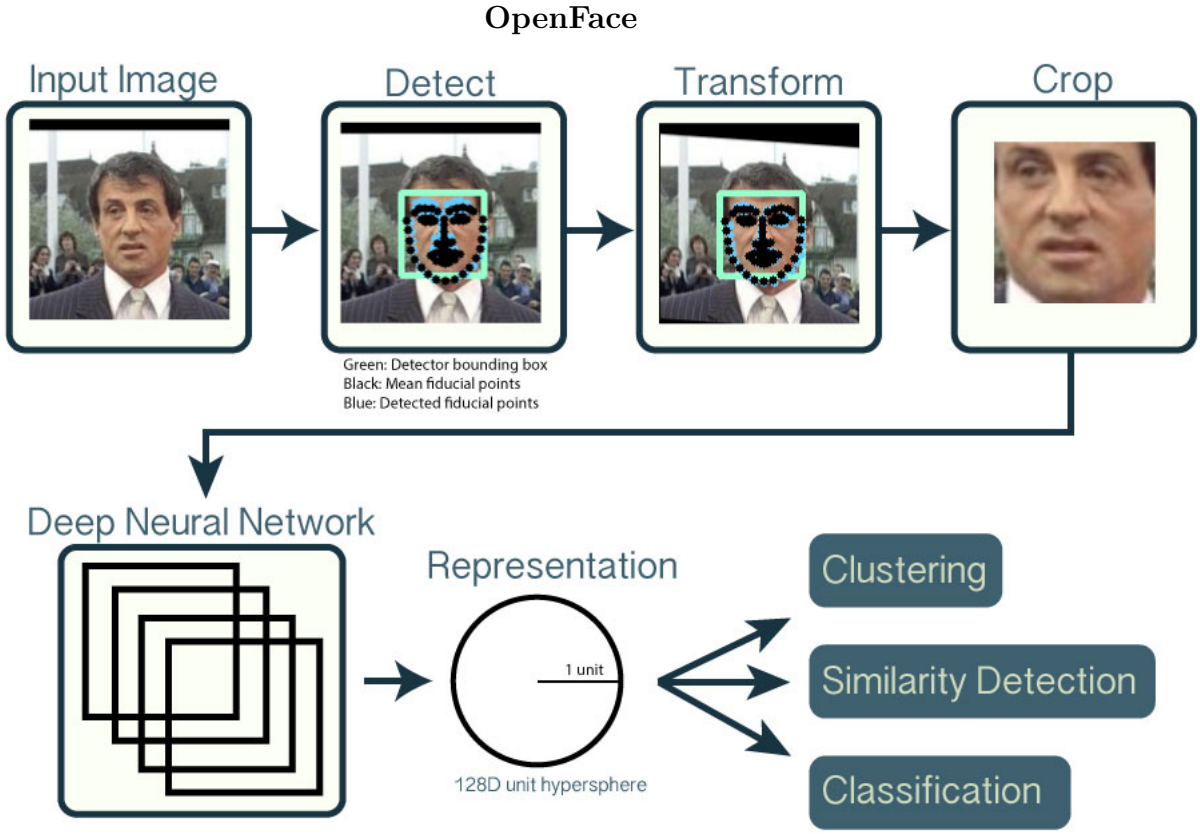


Figura 3: Metodología OpenFace

Para este proceso, se guardará en carpetas información importante. Por ejemplo, guardaremos todas las imágenes tratadas, ya que así, podremos ver qué imágenes han sido rechazadas por no poder leerlas o si tenemos alguna imagen corrupta en nuestro *dataset*. La siguiente información será guardar las imágenes que contienen caras, le aplicamos este filtro respecto al anterior para así ver la diferencia de las imágenes tratadas con las imágenes que contienen caras. Y por último, guardaremos las caras detectadas, es decir, a partir de la *Bounding box* obtenida la agrandaremos para que se aprecie el escenario y recortaremos la imagen, obteniendo así una imagen de la cara y el escenario donde se encuentra.

2.2.2. Face detection pose estimation and landmark localization in the wild

Este algoritmo fue desarrollado en el 2013 por X. Zhu y D. Ramanan para el lenguaje *Matlab*. Este detector facial lo utilizaremos y así, podremos compararlo con *OpenFace* y quedarnos con el que mejor se adecúe a nosotros. Entenderemos su funcionamiento, aunque, es algo complejo, pero, se puede explicar en la forma que, crea unos modelos de árboles para poder detectar caras y las poses en la que se encuentran.

Como veremos en la siguiente figura 4, extraída de la documentación oficial del algoritmo, podemos ver la máscara que utiliza para la detección facial en las diferentes poses que podemos tener una cara en las imágenes.

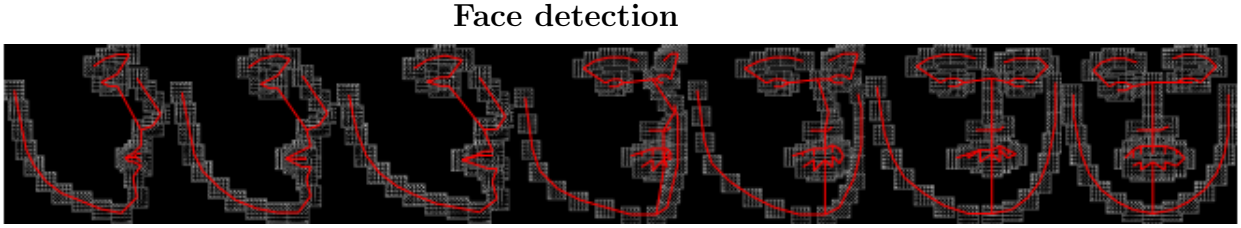


Figura 4: Metodología OpenFace

Como hemos dicho en la sección anterior, buscamos, aparte de encontrar las caras, obtener las características de cada cara, para poder tratarlas y clasificarlas, sin embargo, este algoritmo no nos las facilita, no obstante, nos las localiza y nos dice donde se encuentran mediante una *Bounding box*.

Como ya hemos mencionado, no podremos obtener las características de las caras, pero, necesitamos alguna manera de poder comparar las caras para poder clasificarlas. Entonces, será necesario obtener una matriz de distancias entre las caras. Para ello, utilizaremos un algoritmo llamado Deep-matching, que explicaremos a continuación, mediante los datos extraídos en la *Bounding Box* dada.

2.3. Cálculo de la similitud entre caras

En esta sección, explicaremos los métodos utilizados para poder obtener una matriz de comparación entre las caras, dependiendo del método utilizado para la detección facial, es decir, cuando utilizamos el algoritmo *OpenFace*, tendremos que calcular la matriz de distancia entre las características obtenidas. En cambio, si utilizamos el método *Face Detection*, como ya hemos comentado con anterioridad, tendremos que calcular la similitud entre las caras obtenidas.

2.3.1. Deep - Matching

Deep-Matching es una algoritmo desarrollado por Jerome Revaud en el 2013. Este algoritmo funciona del siguiente modo. De las dos imágenes a comparar, una la utiliza como referencia y la otra como objetivo (target). La imagen de referencia la separa en pedazos (patches), primero en cuatro trozos, luego en ocho y así sucesivamente tantos niveles hasta encontrar el óptimo. En cada pedazo de imagen va comparando con la imagen objetivo.

El funcionamiento de este algoritmo es invisible para nosotros, pues, no necesitamos modificarlo sino, utilizarlo y así obtener la similitud entre las caras encontradas por el algoritmo de *face detection pose estimation and landmark localization in the wild*. Aunque, no tener que modificarlo no significa que no tengamos que conocer

su funcionamiento. Todo lo contrario, debemos conocer como trabaja y así decidir si utilizarlo o no.

Por lo tanto, en nuestro caso, si tenemos una matriz cuadrada donde cada fila y cada columna corresponde a cada cara, al recorrer toda la matriz, tendremos en la diagonal la máxima similitud,sin embargo, no será simétrica puesto que, para que sean simétricas se tendría que dar la casuística de que siempre utilizara la misma imagen de referencia en todas sus comparaciones y, como no es así, nos devuelve una matriz no simétrica de similitudes.

Por todo ello, en la siguiente figura 5, observamos el funcionamiento con dos imágenes, tal y como hemos explicado en los párrafos anteriores.

Face Detector

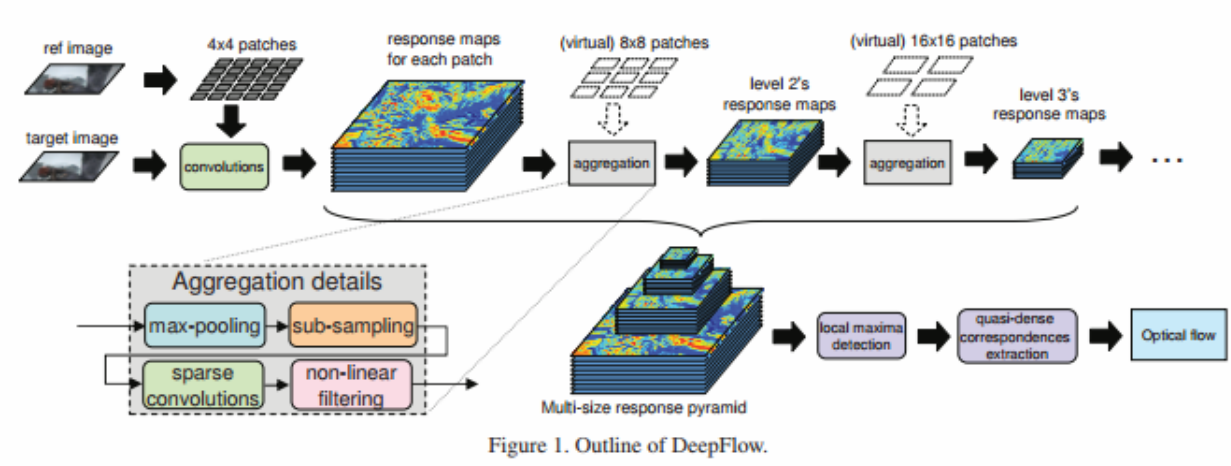


Figura 5: Funcionamiento Deep - Matching

2.3.2. Matriz de distancia

En este paso, necesario para poder hacer algún proceso de aprendizaje y poder así diferenciar a cada persona, calcularemos la distancia entre dos caras. En primer lugar, obtendremos la matriz de características de cada cara, obtendremos dicha matriz al utilizar OpenFace. Una vez las tenemos todas, calculamos la distancia entre las matrices, podemos utilizar diversos métodos.

Aunque, toda distancia debe verificar los siguientes puntos:

1. $d(i, j) > 0$, es decir, no existe la distancia negativa entre dos puntos.
2. $d(i, i) \neq 0$, es decir, la distancia entre dos puntos iguales no puede ser diferente de cero.
3. $d(i, j) = d(j, i)$, es decir, será una matriz simétrica

Una vez cumplen estos tres puntos básicos, estamos delante de una distancia, ahora vamos a ver las diferentes formas de calcularlas.

- **Distancia euclidiana**

$$d(X_1, X_2) = \sqrt{\sum_{i=0}^{N}(X_{1i} - X_{2i})^2}$$

- **Distancia de Manhattan**

$$d(X_1, X_2) = ||X_1 - X_2|| = \sum_{i=0}^{N} |X_{1i} - X_{2i}|$$

En nuestro caso, restaremos las dos matrices de características de cada cara, ambas matrices son de 128 dimensiones, y por último, haremos el producto de las dos matrices, obteniendo así una distancia *d*. Como resultado de esta operación obtendremos una matriz donde las filas y las columnas serán las caras y en la diagonal tendremos la mínima distancia, es decir, 0.0 y será simétrica. Es decir, cumpliremos con los requisitos del cálculo de distancias mencionado anteriormente.

2.4. Métodos automáticos para el clustering de caras

En esta sección, veremos los métodos necesarios para llevar a cabo el aprendizaje automático. Para ello, veremos las diferencias entre aprendizajes supervisados y no supervisados y los diferentes métodos que tenemos dentro de estas dos categorías.

2.4.1. Aprendizaje supervisado

Para este proyecto es difícil poder utilizar aprendizaje supervisado dado que, en un principio, no tenemos las caras con las que se va a interactuar. Pero, para poder entender el funcionamiento del no supervisado, es necesario entender éste.

El aprendizaje supervisado consiste en la introducción de nuevos datos y en base a un entrenamiento previo, tiene que ser capaz de obtener una salida que bien puede ser una etiqueta o un valor numérico. Entonces estaremos hablando de clasificación o regresión.

Clasificación

La clasificación consiste en etiquetar y agrupar los datos de salida, pueden ser categóricas, por ejemplo, agrupar en dos clases, hombres y mujeres, etcétera. Para poder utilizar la clasificación necesitaremos entrenar unos datos. Y a partir del entrenamiento, predecir una salida con los datos de test. La clasificación consiste en, dado una entrada obtener una salida en forma de etiqueta (*label*) y predecir si se trata de una clase u otra. Es decir, en nuestro caso, dado una nueva cara predecir de quien se trata, si ya la conocemos o no.

Regresión

En cambio, la regresión, a groso modo, se utiliza para, dada una entrada, predecir su salida en forma numérica mediante el entrenamiento previo. Y observar la tendencia que sigue.

Es decir, en la regresión lineal, por ejemplo, si tenemos dos variables que dependen directamente entre si, y entramos un nuevo dato al sistema, tendremos que saber posicionarla dentro del sistema.

Ejemplo de regresión

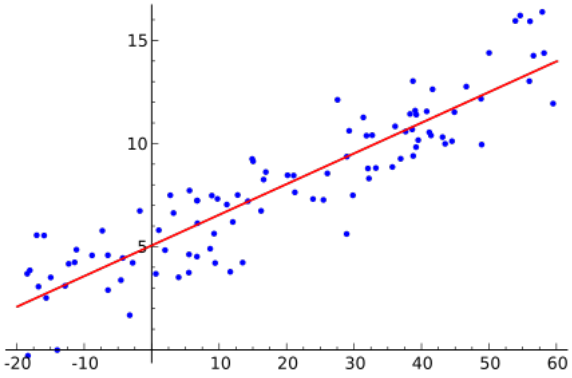


Figura 6: Regresión lineal

Como podemos ver en la siguiente figura (Figura 6), se trata de un ejemplo extraído de *Wikipedia*, donde podemos entender la regresión lineal viendo que sigue una tendencia.

2.4.2. Aprendizaje no supervisado

El aprendizaje no supervisado, a diferencia del supervisado, no necesita unos datos de entrenamiento y observaremos la densidad de los datos, para ello utilizaremos métodos de agrupamiento o *clustering*.

Clustering

En este proyecto se ha decidido utilizar métodos de clustering para poder agrupar las diferentes caras por personas sin la necesidad de tener un conjunto de imágenes para el entrenamiento. Por ello, se ha probado con los diferentes métodos que tenemos disponibles en la librería *scikit learn* de Python. En un principio se pensó utilizar Spectral Clustering para ver su funcionamiento, una vez implementado se pasó a implementarlos otros para ver la diferencia que existen entre ellos.

K-Means

La librería de Python, anteriormente mencionada, permite utilizar k-Means de forma sencilla, pero, para poder utilizar este algoritmo necesitamos introducir el número de clusters, es decir, las personas que interactúan, el problema, mencionado en la sección anterior, es que no disponemos de esa información, a priori. Por ello, tenemos que seguir buscando otros métodos.

Este algoritmo separa k grupos de similar variancia, minimizando la suma de las distancias al cuadrado de cada objeto de un clúster a su centroide μ, esto lo llamaremos inercia.

$$Inercia = \sum_{i=0}^{N} ||x_i - \mu||$$

El objetivo de este método es elegir los k - centroides que reduzcan al mínimo la inercia, para ello, primero se elegirán los centroides para cada uno de los k-clusters. Esto, se puede obtener de dos formas, crear los centroides en el espacio aleatoriamente, o bien, elegir k-objetos del *dataset*, para ello, decimos que hacemos un pre-procesamiento de los datos.

Una vez tenemos los centroides asignados, iremos iterando hasta que converja, o bien, un determinado número de iteraciones. En cada iteración haremos dos tareas, la primera, **asignaremos** cada objeto al centroide más cercano aplicando alguna medida de distancia (euclidiana, de Manhattan, etcétera). El segundo paso, será la **actualización**, calcularemos los nuevos centroides haciendo la media de los objetos que componen un clúster.

En la siguiente imagen podemos observar el funcionamiento de K-Means, tal y cómo se ha explicado.

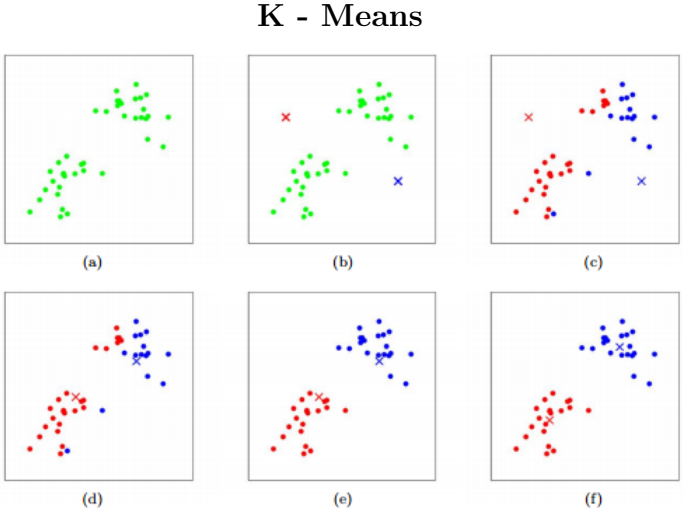


Figura 7: Funcionamiento K - Means

Hierarchical clustering

También llamado *hierarchical clustering análisis (HCA, por sus siglas en inglés)* busca construir una jerarquía de agrupaciones. Para ello, podemos ver dos formas de hacerlo.

- **Agglomerative**
- **Divisive**

Agglomerative es un algoritmo iterativo el cual, cada nodo será un *clúster* y en cada iteración se irán agrupando los *clústers* con mayor similitud, esta similitud la podemos ver como la menor distancia entre *clústers*

En cambio, *Divisive*, es todo lo contrario, es también un método iterativo, sin embargo, al inicio de la ejecución, tendremos un solo *clúster*, el cual se irá descomponiendo.

La pregunta que nos tenemos que hacer es, "¿Cuándo se paran estos dos algoritmos?", pues esta pregunta no es fácil de responder y tendremos que ver donde poner el umbral para detenerse.

Para este proyecto, trabajaremos con agglomerative clustering, como entrada tenemos la matriz de distancia de cada cara. Entonces, cada cara será, al principio del algoritmo, un *clúster*, así, tendremos inicialmente, tantos *clústers* como caras tenga nuestro *dataset*. Irá iterando hasta un determinado umbral. La salida la podemos representar con un dendograma.

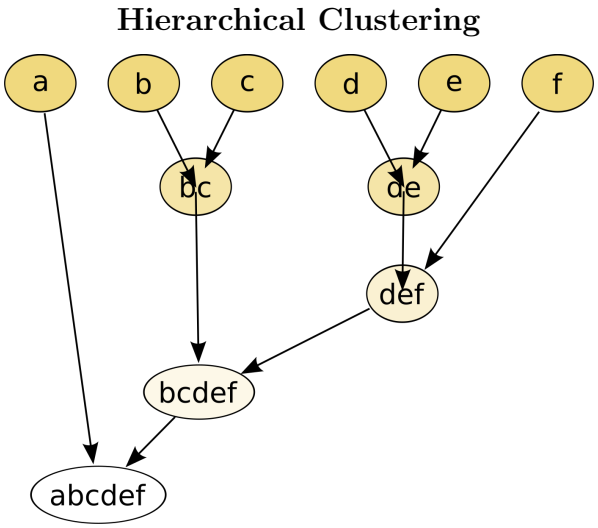


Figura 8: Hierarchical Clustering en modo Agglomerative

Mean Shift

Este algoritmo es un método iterativo el cual encuentra las modas de unas distribuciones pero, sin la necesidad de saber el número total de modas que tenemos en nuestro *dataset*. Por tanto, considera que el espacio de datos es una función de densidad, y es por ello, que para cada dato encuentra su moda más cercana. En definitiva, define una región alrededor de cada punto y encuentra su media, cambiando la situación de su media actual a su media. Este procedimiento se repite hasta que converja.

La forma de calcular la media se hace mediante lo que llamamos kernel. Este algoritmo nos resulta especialmente interesante ya que no necesitamos indicarle el número de clústers que necesitamos.

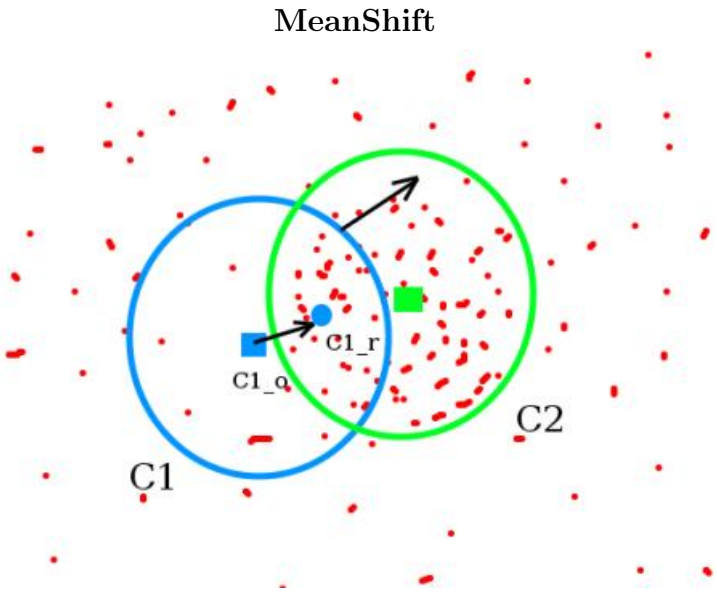


Figura 9: Mean Shift, funcionamiento

Spectral Clustering

Este método, toma como entrada la matriz de distancia, hace una reducción de dimensionalidad para poder tratar los datos en dos dimensiones.

El objetivo de *Spectral clustering* es obtener clústers de los datos conectados pero no necesariamente compactados.

Ward

El método de clustering de Ward es un método jerárquico como ya hemos visto. Se trata de ir agrupando e ir haciendo clústers pero, para poder juntar dos clústers Ward propone juntarlos siempre que cumplan el valor óptimo de una función objetivo, por ejemplo, puede ser el valor del error de la suma de los cuadrados.

DBSCAN

El agrupamiento espacial basado en densidad de aplicaciones con ruido encuentra los números de clústers según la densidad de los conjuntos, como entrada deberemos indicar una epsilon y el número puntos que consideremos denso. Este método resulta muy bueno para este proyecto ya que no necesitamos introducir el número de clústers cómo ocurre con K-Means. Pero, como desventaja, si tenemos unos datos muy compactos será difícil poder obtener buenos clústers. Por ello, es bueno tener unas buenas características para cada cara y escoger un buen método para calcular la distancia entre cada par de imágenes.

Ejemplo de DBSCAN

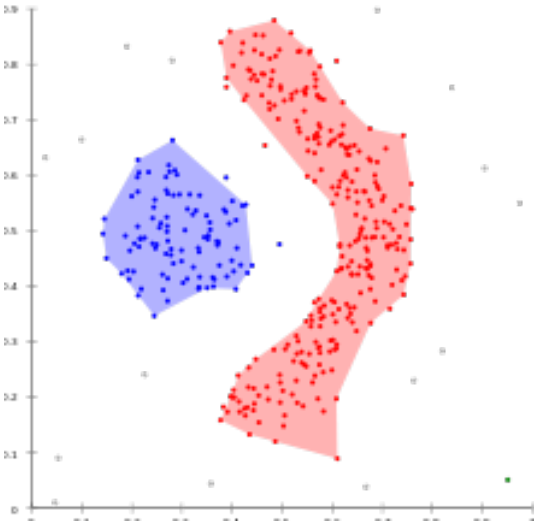


Figura 10: DBSCAN

Birch

Y por último, veremos este método de clustering, Birch es un algoritmo de la familia de los jerárquicos, el cual, no es necesario indicarle el número de clústers que queremos obtener.

Está muy bien indicado para este proyecto ya que trabaja muy bien para dimensiones altas, por lo tanto, le podremos pasar las características de las caras que, recordemos que tienen 128 dimensiones.

2.5. Visualización de los resultados obtenidos

En este apartado nos centraremos en la visualización de las personas que previamente hemos agrupado. Para ello, desarrollaremos una interfície gráfica con el modulo *Tkinter de Python*.

En primer lugar, ordenaremos las imágenes por el momento de la captura de la imagen y para tal propósito, será necesario acceder a los *metadatos* de las imágenes de nuestro *dataset*. En segundo lugar, daremos la opción al usuario a elegir la persona que quiere visualizar y analizar las horas con más interacciones con la persona escogida. Y por último, una vez seleccionada la persona que se quiere visualizar se podrá saber que días se ha interactuado mediante un calendario.

A continuación, exponemos los métodos para poder mostrar las caras en una gráfica, para ello, hacemos una reducción de dimensionalidad la cuál mostramos los dos métodos que hemos visto.

2.5.1. Reducción de la dimensionalidad

En este apartado veremos la importancia que tiene la reducción de dimensionalidad, ya que, nos resulta imposible poder visualizar todas las características de las caras encontradas en uan gráfica ya que disponemos de una dimensionalidad elevada.

Para poder mostrar los resultados en una gráfica y poder trabajar con los clústers de forma más sencilla necesitamos hacer una reducción de la dimensionalidad. En concreto, tenemos las características de *OpenFace* las cuales tienen una dimensión de 128, y obviamente, no podemos dibujarlas en una gráfica y poder interpretarla fácilmente. Para ello, tenemos dos métodos muy efectivos, los cuales, nos hablan de ellos en la documentación de *OpenFace*, en particular, nos habla principalmente sobre t-SNE, pero, encontramos PCA para poder hacer la comparativa.

En general, el proceso buscamos es, teniendo un conjunto de características, en este caso de 128 dimensiones, reducir esta dimensionalidad por una que podamos visualizar en una gráfica y podamos entender, es decir, 2 o 3 dimensiones.

2.5.2. PCA

El método del análisis de componentes principales (PCA, por sus siglas en inglés) busca la reducción de la dimensionalidad y trabaja para que los datos queden representados en términos de mínimos cuadrados.

Se trata de convertir un conjunto de posibles variables correlaciones en variables sin correlación lineal y esto se llama componentes principales.

En nuestro proyecto y dado que tendremos una gran dimensionalidad en las características de las caras, necesitaremos un algoritmo como este para poder mostrarlo.

2.5.3. t-SNE

El t-SNE (t-distributed stochastic neighbor embedding) es un método no lineal para la reducción de dimensionalidad. Este algoritmo sigue dos estrategias principales. La primera es construir una distribución de probabilidades sobre cada par de objetos que tenemos en la matriz de alta dimensión. Y en segundo lugar, define una distribución de probabilidades similar a la anterior pero, con la diferencia de hacerlo sobre el conjunto con una baja dimensionalidad.

3. Pruebas y resultados

En este apartado clasificaremos todas las pruebas que hemos ido realizando durante el desarrollo de este proyecto, así como los resultados obtenidos y unas comparaciones sobre los resultados esperados y los obtenidos.

Empezaremos explicando los pasos realizados para cada uno de los apartados anteriormente mencionados.

Para ponernos en situación, hay que explicar que, en un principio se pensó diseñar todo el proyecto en *Matlab*, y se desarrolló hasta el punto de calcular la matriz con Deep-Matching, puesto que los resultados obtenidos no eran los esperados, se buscó un nuevo algoritmo para la detección facial y con ello surgió la idea de implementarlo con *Python* y así utilizar *OpenFace*.

Con este nuevo algoritmo, teníamos mejor precisión para el reconocimiento facial y así poder obtener mejor resultados. Cabe decir que, la documentación, al ser un algoritmo nuevo, es escasa, pero *Python* ofrece las mismas oportunidades que *Matlab* y además, es *Open source*.

3.1. Creación del *dataset*

El *Ground Truth* consiste en procesar un conjunto de datos con el fin de poder evaluar los diferentes algoritmos utilizados, es decir, para la detección facial, el *Ground Truth* consistirá en indicar, para cada imagen, dónde se encuentra las diferentes caras, en una *Bounding box*. En primer lugar, tendremos que analizar nuestro *dataset* e ir seleccionando las caras de cada imagen, para ello, utilizaremos un código para *Matlab*, en el cual, indicaremos la esquina superior izquierda y la esquina inferior derecha, creando así un rectángulo y dentro se debe encontrar una cara. Tendremos que hacerlo para todas las caras de todas las imágenes. Esto nos llevará bastante tiempo pero, es muy valioso ya que con ello, podremos analizar la precisión del detector de caras utilizado.

Ground Truth para clusters

En este apartado hablaremos sobre la evaluación de los clústers creados, para ello, será necesario tener cada persona en una carpeta, así sabremos de forma empírica cuantas personas tenemos en nuestro *dataset*. En esta carpeta contendrá todos los *Bounding boxes* donde aparece una persona en nuestro *dataset*, para ello, será necesario tener el *Ground Truth* inicial, explicado en la sección 4.2.

En primer lugar, en nuestro *dataset* tendremos, para cada secuencia, un archivo para cada persona que aparece, y que contiene la *Bounding box* de cada imagen. Entonces, lo que tendremos que hacer será poner cada cara diferente guardar ese archivo en carpetas.

Este proceso nos llevará bastante tiempo, ya que nuestro *dataset* es grande y aparecen muchas personas diferentes, pero es muy útil y necesario hacerlo. Al final

del proceso, deberemos tener cada persona en una carpeta, en forma de ficheros donde se encuentra las *Bounding box* para cada secuencia.

Una vez tenemos los pasos anteriores, crearemos un fichero Excel que contendrá las imágenes ordenadas por secuencia y a su vez cada imagen tendrá la *Bounding box* que clasificamos y ésta a su vez, asociada al *clúster* correspondiente. Este fichero no es necesario para la aplicación, pero, si que es muy útil para nosotros mismos.

<Secuencia> <ID_imagen> <atributos_bounding_box> <clúster>

Los atributos de la *Bounding box* serán x, que es el punto en el eje de las equis. El punto y, en el eje de las íes. El tercer atributo es w que significa el ancho de la *Bounding box* y por último, tenemos el h que será el largo de la *Bounding box*.

3.2. Detección facial

Empezaremos con la detección facial, como hemos mencionado con anterioridad, hemos utilizado dos algoritmos diferentes, uno con Python (OpenFace) y otro con Matlab (FaceDetection).

Para *OpenFace*, en primer lugar, hemos aprendido el funcionamiento mediante las explicaciones encontradas en su página web, podremos encontrar varios ejemplos y probar. Más tarde, adaptaremos el código de ejemplo para nuestro caso. Esto, lo dividiremos en dos partes. Una, será la detección de la cara en cada foto, para ello, utilizaremos una función llamada *getAllBoundingBoxes* la cual nos retornará todas las *Bounding boxes* encontradas para una foto, a nosotros solamente nos interesan las personas que puedan estar interactuando, por ello, deberemos seleccionar los *Bounding boxes* que sean más grandes que un cierto tamaño, es la única forma que tenemos para saber que la persona está cerca, si su *Bounding boxes* es grande. Descartaremos todas aquellas personas que aparezcan alejadas.

También modificaremos el tamaño de la *Bounding boxes* de cada cara, creemos que es importante tener las características de las caras pero también observar el entorno. Para ello, agrandaremos la *Bounding boxes* 30 puntos sumados a los que nos detecta *OpenFace*. Un segundo paso será obtener las características de cada cara, para ello utilizaremos el framework,instalado previamente, *torch*, esto será necesario para cada cara encontrada. Nos retornará una matriz con 128 dimensiones, que serán las características para aquella cara. Y las guardaremos, cada una de ellas, en un fichero.

3.3. Evaluación de la detección facial

Una vez obtenido el Ground Truth, podremos calcular los *True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN)*, y con ello la precisión, el recall y el F-Measure. Pero vayamos por partes, identificaremos que es cada término.

- **True Positive (TP):** Diremos que tenemos un TP cuando nuestro algoritmo detecte una cara y, efectivamente, con nuestro Ground Truth hayamos indicado que existe una cara.
- **True Negative (TN):** Consideraremos este caso cuando, el algoritmo utilizado de detección facial no detecte cara y no existan caras.
- **False Positive (FP):** En este caso, tendremos un FP siempre y cuando, el algoritmo utilizado detecte cara y no exista cara en esa imágen.
- **False Negative (FN):** Y por último, diremos que tenemos un FN cuando el algoritmo utilizado no detecte cara, sin embargo, nosotros, por nuestro Ground Truth, sabemos que sí que la hay.

Para poder obtener estos cálculos, será necesario comparar la *Bounding boxes* creada con nuestro Ground Truth con los rectángulos obtenidos por el algoritmo utilizado para la detección facial. Es decir, tendremos que calcular el área entre los dos rectángulos y si supera un umbral, determinado previamente por nosotros, tendremos un TP.

Cálculo del área

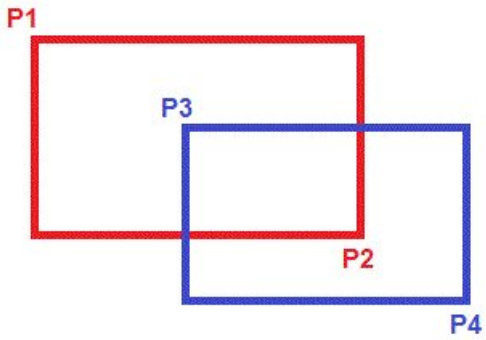


Figura 11: Calculo del área del rectángulo interior

Como podemos observar en la figura 11, si imaginamos que el rectángulo de color rojo es el obtenido haciendo el Ground Truth para una determinada imagen, y el rectángulo azul es el resultado de la cara encontrada por nuestro algoritmo, entonces, el rectángulo creado entre *P3* y *P2* debe ser lo suficientemente grande para poder decir que, efectivamente, tenemos un TP.

Si por el contrario, tenemos rectángulo rojo y azul, pero la intersección es tan pequeña que no lo consideramos TP, tendremos un False Negative, ya que sabemos que existe una cara pero el algoritmo utilizado no lo ha encontrado y también tendremos un False Positive ya que ha detectado una cara cuando, por el umbral, no la hay.

Otro caso que podemos encontrar es que solamente tengamos el rectángulo rojo, recordemos que el rojo viene dado por nuestro Ground Truth, y por tanto, en este caso nos encontramos con un False Negative, ya que el algoritmo utilizado para la detección facial no ha sido capaz de encontrar la cara.

Y por último, se puede dar el caso de que, solamente tengamos el rectángulo azul, es decir, que el algoritmo utilizado haya detectado una cara cuando en realidad, sabemos que entre esos puntos no se encuentra ninguna cara.

Una vez tenemos todos los *True Positive, True Negative, False Positive y False Negative* es hora de calcular la precisión, recall y F-Measure.

$$Precision = \frac{TP}{(TP + FP)}$$

Con la fórmula de la precisión obtendremos las detecciones correctas, es decir, las que detecta cara y efectivamente existe una cara entre ese rectángulo y todas las detecciones de nuestro algoritmo, las que existen caras y las que no.

Otra expresión que deberemos evaluar será el *recall*

$$recall = \frac{TP}{(TP + FN)}$$

Con esta fórmula, obtendremos los elementos, en nuestro caso, las caras, relevantes, es decir, obtendremos las caras detectadas por el algoritmo utilizado y las caras que existes pero que el método de detección facial no ha sido capaz de localizar.

A continuación, calcularemos la expresión F-Measure, también conocido como *F-score*, es una medida para calcular la exactitud de nuestro sistema. Para ello, utilizaremos la siguiente expresión:

$$F - Measure = \frac{2 \times TP}{(2 \times TP) + FP + FN}$$

O, otra opción para calcular puede ser,

$$F - Measure = 2 \times \frac{precision \times recall}{(precision + recall)}$$

En la figura 12, extraída de *Wikipedia*, podemos ver de forma gráfica, como calcular las expresiones comentadas anteriormente.

Gráfico sobre precisión y *recall*

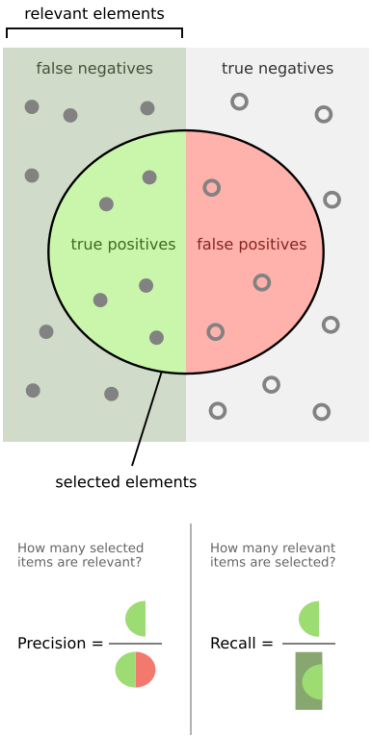


Figura 12: Cálculo de precisión y *recall*

Una vez tenemos todas las fórmulas necesarias y entendiendo bien todas las terminologías procedemos a comparar los dos métodos utilizados. Así podremos comparar y demostrar de forma empírica, que algoritmo funciona mejor para nuestro *dataset*.

	Algoritmos utilizados	
	OpenFace	FaceDetection
True Positive	**0.545**	**0.277**
True Negative	**0.309**	**0.433**
False Positive	**0.066**	**0.093**
False Negative	**0.078**	**0.195**
Precision	**0.892**	**0.747**
Recall	**0.874**	**0.587**
F-Measure	**0.883**	**0.657**

Cuadro 1: Tabla comparativa con los dos algoritmos utilizados

En la tabla 1, podemos ver la comparativa entre los dos métodos utilizados con el sistema que hemos explicado para poder contabilizarlo. Vemos que OpenFace nos da unos resultados un tanto superiores y dado que, es *Open source* con *Python* y haciendo una valoración general, creemos que debemos utilizarlo.

Todos los resultados se irán guardando en ficheros de *log* y podremos tener un histórico sobre los diferentes *datasets* utilizados con los algoritmos mencionados.

3.4. Búsqueda de la similitud entre caras

3.4.1. Matriz de distancias

Una vez tenemos todas las características es hora de calcular las distancias entre cada cara, es decir, las distancias entre las diferentes características. Para ello, tendremos varias formas de calcularlo, nosotros calcularemos la distancia restando las dos matrices de características y con el resultado haremos el producto de las matrices, es decir, para cada par de matrices de características obtendremos un valor mayor que cero. En la diagonal tendremos la distancia mínima, es decir, cero.

Tenemos un ejemplo muy claro donde se puede ver que la distancia no es perfecta, por ejemplo, *OpenFace* nos recomienda en su documentación que dos caras pueden ser la misma persona si la distancia, calculada como hemos explicado con anterioridad, es menor a 0.99, bien pues con el siguiente ejemplo (Figura 6) podemos observar que no siempre es igual.

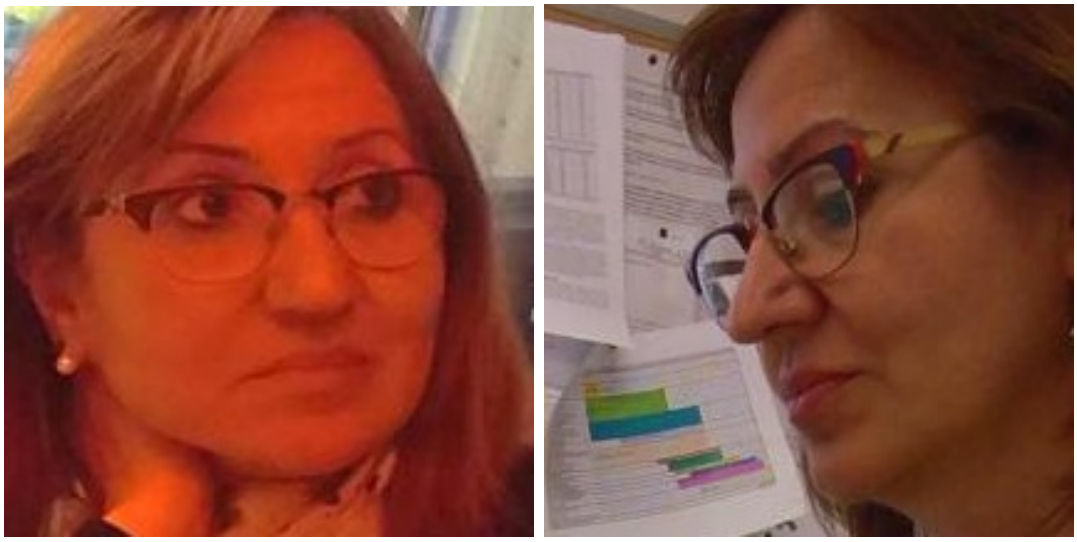
(a) Cara 1 (b) Cara 2

Figura 13: Caras diferentes.

Como podemos observar no se trata de la misma persona, pero, la distancia calculada nos da una una distancia de **0.811**, esto se podría resolver fácilmente observando que distancia acierta más sobre un conjunto aleatorio de nuestro *dataset*, el problema viene al ver que con otro par de caras, esta vez iguales, nos encontramos que obtenemos una distancia superior a la anterior, es el caso siguiente (Figura 13).

Para el siguiente caso (figura 14), la distancia obtenida con *OpenFace* es de **1.817**, como consecuencia de este hecho, el clustering se dificulta, y hace que sea más difícil poder agrupar a las personas iguales.

También, para hacerlo más visual, creamos un archivo Excel, a partir, de la matriz por si queremos ver de forma sencilla la matriz de distancias creada.



(a) Cara 1 (b) Cara 2

Figura 14: Caras diferentes.

Por tanto, la matriz que obtendremos serà del tipo:

	A	B	C	D
A	**0.0**	d(b,a)	d(c, a)	d(d, a)
B	d(a,b)	**0.0**	d(c, b)	d(e, b)
C	d(a,c)	d(b,c)	**0.0**	d(d, c)
D	d(a,d)	d(b,d)	d(c,d)	**0.0**

Cuadro 2: Ejemplo matriz de distancia

Siendo *A, B, C y D* las caras encontradas por el algoritmo que hemos utilizado, y siguiendo las reglas mencionadas con anterioridad vemos que se trata de una matriz cuadrada, en la diagonal encontramos la distancia mínima entre dos caras, es simétrica y las distancias no son menores que cero.

3.4.2. Deep - Matching

Las pruebas realizadas con Deep-Matching han sido para comprobar y poder extraer la matriz de similitud. Cabe decir, que este algoritmo no nos dará la menor distancia entre dos imágenes, sino que en la diagonal tendremos la máxima similitud, es decir, tendremos lo contrario a la matriz de distancia.

Para ello, iremos recorriendo las imágenes y llamaremos a la función de Deep-Matching para cada par de imágenes e iremos guardando el resultado en un fichero *.mat*.

3.5. Agrupaciones por personas

Clustering

En esta sección nos centraremos en explicar las pruebas y los algoritmos elegidos para hacer clustering. En primer lugar, hay que explicar que para llegar a este punto tenemos que estar seguros de que los pasos anteriores (*secciones 4.1 y 4.2*) están correctamente implementados. Para ello, tendremos un juego de pruebas para poder verificarlo de forma sencilla. Obtendremos una o dos secuencias de nuestro *dataset* para hacer las pruebas, en ella habrán aproximadamente unas diez fotografías, en las que verificaremos que hayan imágenes con una o más caras e imágenes sin cara. De esta forma, podremos comprobar el detector de cara y las distancias.

Una vez tenemos las características de cada cara y la matriz de distancia es hora de buscar e informarnos de que método es más idóneo para nosotros.

El problema

Nos encontramos con unas imágenes las cuáles tenemos que predecir cuántas personas aparecen en ellas de forma totalmente autónoma. Para ello, y antes de empezar a programar, investigamos sobre los métodos existentes, a grandes rasgos, tenemos que saber y entender la diferencia entre aprendizajes supervisados y no-supervisados. El aprendizaje supervisado necesita un entrenamiento a partir de los datos para predecir la salida. La idea de nuestro proyecto es poderla utilizar para varios pacientes, que estos a su vez tendrán interacciones con diferentes personas. Por ello, creemos que lo mejor es utilizar aprendizaje no-supervisado. Por tanto, tendremos un conjunto de variables las cuales, ya sea por la matriz de características o por la de distancia, deberemos predecir, sin entrenamiento previo, cuantas personas tenemos en nuestro *dataset*.

Elección de los diferentes algoritmos

En este parágrafo, explicaremos los algoritmos elegidos y su motivo. Para ello, primero buscaremos información sobre que algoritmo utilizar. La primera idea es utilizar uno que no sea necesario indicarle los clusterings inicialmente, pues no los conocemos, por ello, k-Means queda descartado en un primer momento, ya que es totalmente necesario introducir el número de clústers inicialmente.

Vemos que los algoritmos jerárquicos se adaptan a nuestras necesidades, pues buscan las agrupaciones mediante dos formas generalmente. Una es ascendente y la otra descendente. Utilizaremos para este trabajo la forma ascendente, es decir, para nosotros, cada cara será un clúster en el inicio del algoritmo e irán agrupándose mediante las distancias. Con ello se creará un dendograma, y podremos observar de forma visual el resultado. Con el fin de utilizar este algoritmo, será necesario como entrada la matriz de distancia simétrica. Éste, nos devolverá unas etiquetas que serán las agrupaciones oportunas. No es necesario introducir el número de clústers, sin embargo, será necesario indicar un umbral. Dicho umbral, nos dará el

corte necesario para separar las imágenes, sino lo indicáramos generaríamos un sólo clúster.

A continuación, calcularemos la precisión del método utilizado. Este paso es muy útil para cuantificar la calidad del método utilizado y para ello, utilizaremos una librería de Python llamada *normalized mutual score*, la cual le pasaremos las carpetas que hemos obtenido con el método de clustering y el Ground Truth, que hemos creado y explicado con anterioridad en la sección 3.1. Esto, nos devolverá una puntuación de cero a uno, donde cero nos indica que son totalmente distintos y uno que son totalmente iguales. En nuestro caso, si tuviésemos un uno, podríamos decir que tenemos un clúster puro.

Dicho esto, evaluamos el método *aglomerativo jerárquico* utilizado para las agrupaciones y obtenemos un resultado de **0.26**, es decir, que tenemos un clúster con un acierto de un 26 %. Dado que es una cifra muy baja, buscaremos otros sistemas para que nos puedan dar mejores resultados. Para ello, probaremos la clasificación.

Resultados de los diferentes métodos de clustering

```
The result of evaluation of the Birch is 0.0176
The result of evaluation of the AffinityPropagation is 0.0329
The result of evaluation of the SpectralClustering is 0.0295
The result of evaluation of the DBSCAN is 0.0285
The result of evaluation of the MiniBatchKMeans is 0.0192
```

Figura 15: Resultados de los diferentes métodos de clustering

Como podemos observar en la figura 15, los resultados que obtenemos son verdaderamente bajos, por ello, seguimos buscando otras alternativas que nos puedan dar soluciones mejores para nuestro proyecto.

Clasificación

En este apartado explicaremos la clasificación realizada con nuestro *dataset*. En un principio, no se pensó en utilizar aprendizaje supervisado ya que no sabemos las personas que nos podrán entrar en nuestro sistema, pero, es un buen método para poder hacer una comparativa y ver si obtenemos mejores resultados. Por tanto, vamos a seguir las instrucciones de *OpenFace* para la clasificación que propone.

En primer lugar, deberemos preparar nuevamente nuestro *dataset* para preparar el conjunto de entrenamiento que utilizaremos para la clasificación. Para ello, será necesario que tengamos una estructura donde cada persona corresponda a cada carpeta, es decir, en cada carpeta debe haber imágenes de la misma persona y deben tener extensión *JPG o PNG*, el nombre de la imagen será indiferente pero, si conocemos el nombre de la persona podemos ponerlo de nombre de la carpeta y así cuando obtengamos resultados veremos de quien se trata. En nuestro caso, desconocemos el nombre de las personas que aparecen en nuestro *dataset*, por tanto, les pondremos un número.

El resultado del primer paso obtendremos la estructura que podemos ver en la figura 16.

Estructura para la clasificación

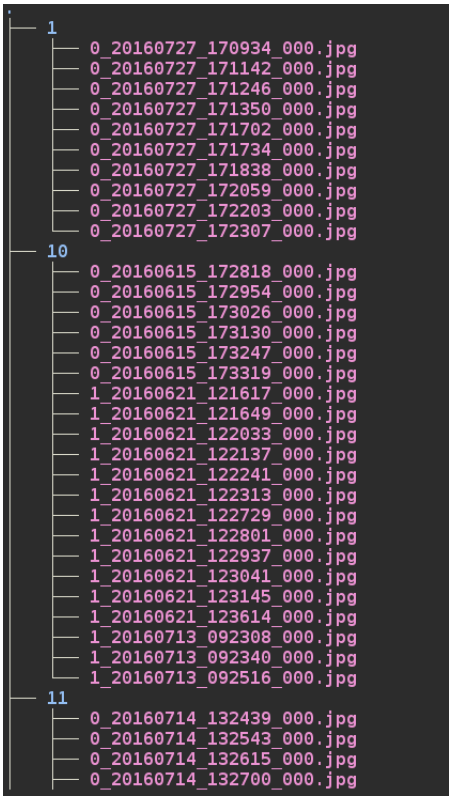


Figura 16: Estructura necesaria para la clasificación

Para los directorios que no tengan muchas imágenes para el entreno, es decir, que tengamos pocas fotos de una persona se puede eliminar indicando el umbral, en nuestro caso será los directorios que tengan menos de tres imágenes.

El siguiente paso, será generar las representaciones para ello habrá que indicar el directorio donde se guardaran las características y las etiquetas.

El próximo paso, será crear el modelo de clasificación el cual se hace mediante un *Support Vector Machine*. Y por último, una vez tengamos los pasos anteriores, nos habrá creado un fichero con extensión *pkl*, por tanto, podremos añadir una nueva imagen al sistema y este nos predecirá en que clase corresponde.

Una vez tenemos la clasificación para nuestro *dataset* y hemos introducido todas nuestras imágenes en el sistema. Es hora obtener, mediante el sistema de evaluación que hemos creado, la evaluación de esta clasificación. Para ello, calcularemos la similitud entre nuestro Ground Truth y la clasificación realizada. Este proceso lo realizamos con una librería de Python llamada *normalized mutual score*. Y el resultado es **0.70**. Es decir, tenemos un acierto del 70 %. Un resultado más elevado que el que hemos obtenido mediante el aprendizaje no supervisado. Por tanto, podemos concluir que, para nuestro caso, nos funcionará mejor un sistema supervisado.

3.6. Visualización

En este apartado explicaremos la interfície gráfica creada para poder visualizar las diferentes personas encontradas con los métodos explicados anteriormente.

Para ello, utilizaremos la librería *Tkinter* de Python, es una librería muy fácil de utilizar y con muchos recursos disponibles, a parte de una extensa documentación en su página web.

Dicho esto, buscaremos una forma de visualizar sobre un calendario que días se a interactuado con una persona. Para ello, será necesario leer los metadatos de las imágenes y obtener la fecha de creación. Hay imágenes que no tienen esta información, si esto ocurre, cogeremos la fecha de modificación. Una vez pintamos los días que ha habido interacción con la persona que hemos seleccionado, miramos otras formas de evolucionar la interfície gráfica. Una idea es visualizar las imágenes que corresponden a la misma persona, ordenadas por fecha.

Interfície gráfica

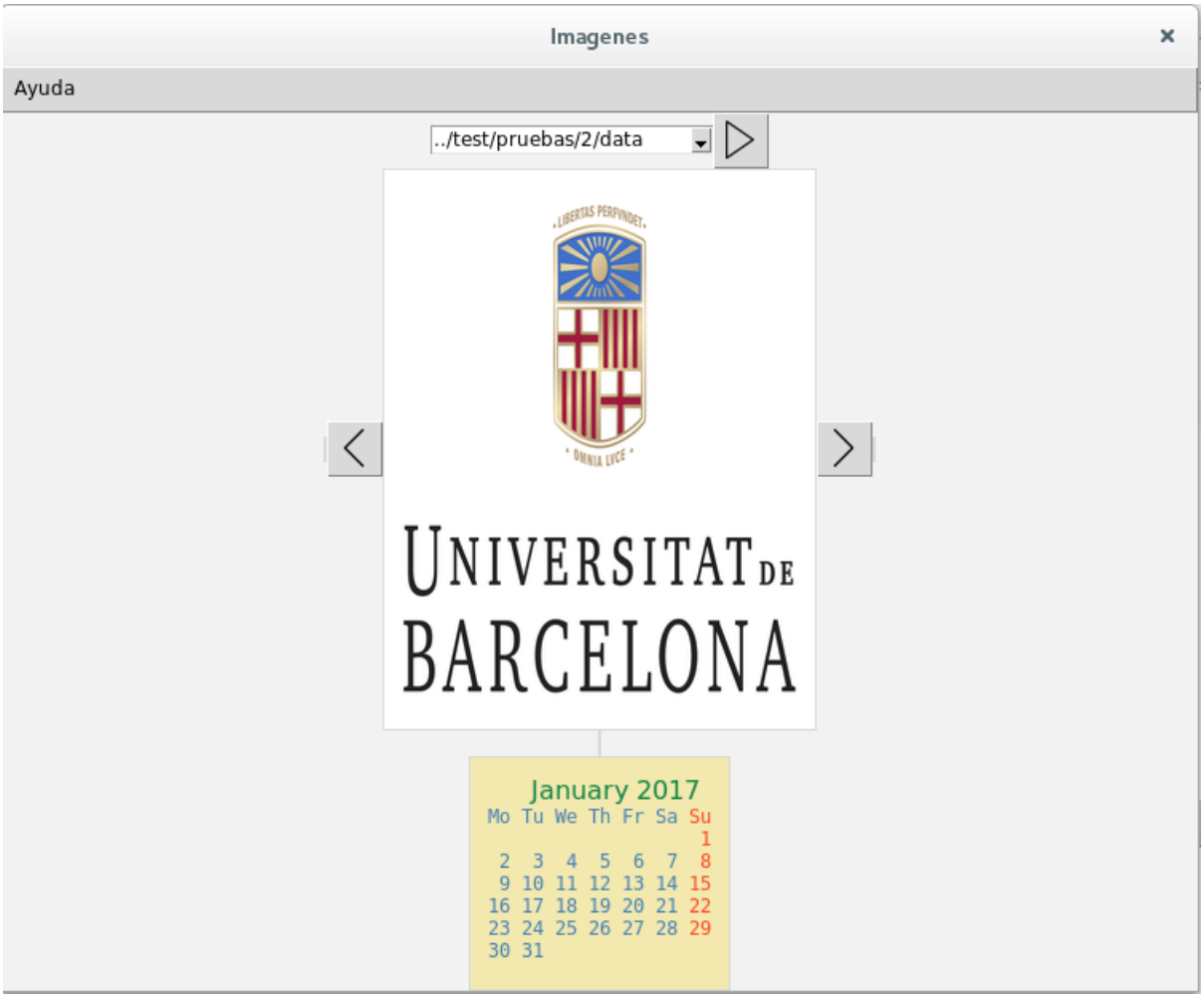


Figura 17: Inicio de la interfície gráfica

Como podemos ver en la figura 17, cuando iniciemos la aplicación veremos un desplegable en la parte superior para poder elegir la carpeta donde se encuentra la persona la cual queremos ver su interacción con el portador de la cámara y, justo a la derecha el botón para iniciar la visualización.

Una vez iniciamos la visualización, podremos ver los días que se ha interactuado con la persona seleccionada mediante el calendario que se muestra abajo. Los días de interacción se mostrarán marcados de color verde y podremos ver las imágenes donde aparece dicha persona.

3.6.1. Reducción de dimensionalidad

La reducción de dimensionalidad nos resultará muy cómoda para poder visualizar nuestros datos, tal y como hemos explicado con anterioridad.

Realizaremos pruebas con dos métodos diferentes. Como podemos ver en la siguiente figura (18) se trata del algoritmo t-SNE, el cual entrena los datos y los reduce, son aproximaciones y si volvemos a ejecutarlo con las mismas características la gráfica variará.

Reducción de los datos con t-SNE

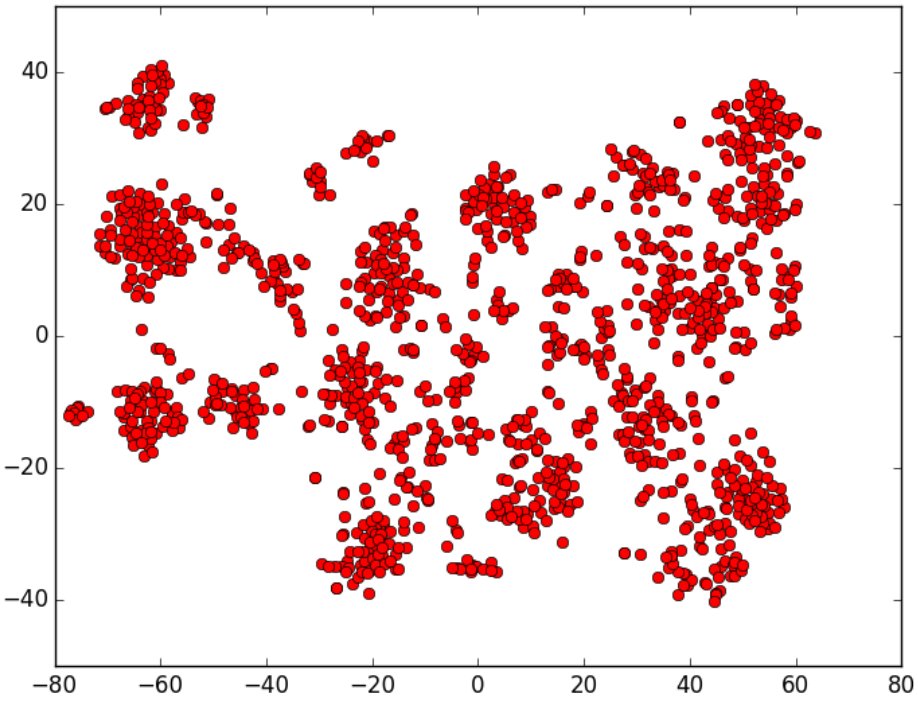


Figura 18: Reducción de los datos utilizando t-SNE

El siguiente método es el llamado PCA, este método es mucho más rápido que el anterior pero los resultados obtenidos, en este caso, cuestan más de entender.

Reducción de los datos con PCA

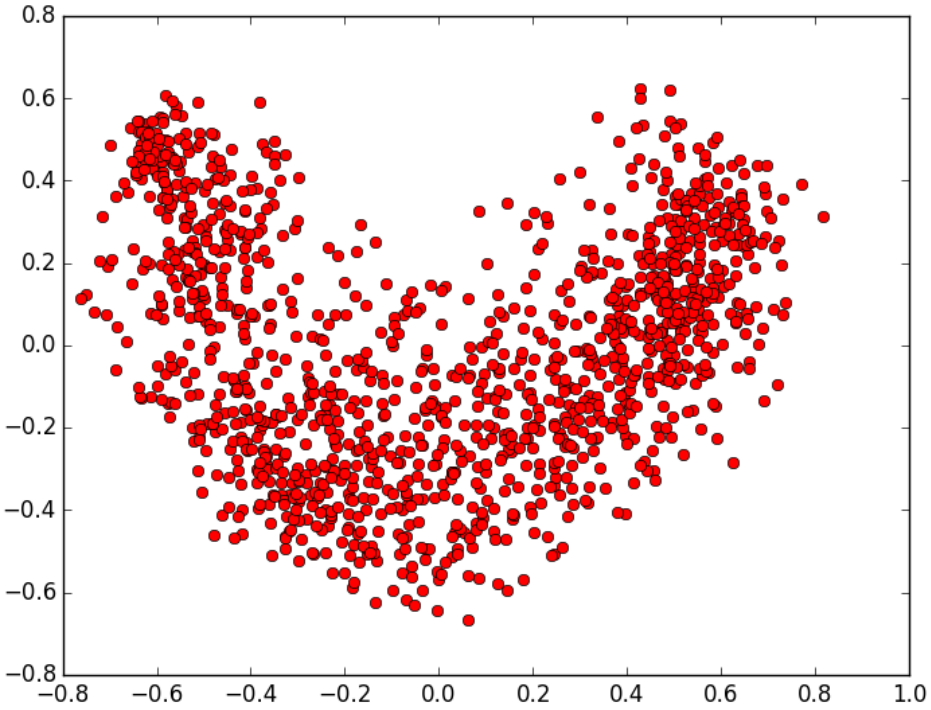


Figura 19: Reducción de los datos utilizando PCA

Para obtener estos resultados, en primer lugar, leeremos los ficheros que tenemos guardados en la carpeta de características (*features*), obviamente, antes tenemos que obtener las características de las imágenes. Una vez leídas todas las características, lo guardaremos todas juntas en otro fichero. Más tarde, ejecutaremos el método que deseemos (*PCA o t-SNE*). Una vez ejecutamos lo dibujaremos en la gráfica.

El hecho de guardarlo en ficheros nos sirve para, más tarde, al utilizar el clustering poder dibujarlo y esta vez poner colores a cada clúster.

4. Estructura del código

En este apartado se explicará la lógica de la aplicación. Está desarrollado en *Python*, aunque también tenemos algún pequeño proceso para *Matlab*, que a continuación explicaremos.

Para poder ejecutar la aplicación leer anexo A (Instalación del código). Una vez tengamos instalado todo el software necesario podremos ejecutar nuestra aplicación. En primer lugar, y antes de ejecutar el código, deberemos configurar el archivo *.ini*, indicándole la ruta de las carpetas donde se encuentran las imágenes. Solo será necesario indicar la carpeta donde se encuentran las imágenes que queremos analizar y estás se deben encontrar dentro de una carpeta, que contenga una sub-carpeta con las secuencias a analizar y las imágenes deben tener la extensión jpg.

Una vez ejecutamos, nos aparecerá un menú de consola, en él podremos hacer las siguientes funcionalidades.

1. Calcular las características de las imágenes.
2. Calcular la distancia de las caras
3. Realizar una reducción de dimensionalidad
4. Obtener los resultados de los clusters
5. Calcular la accuracy y el recall

En este menú, un paso depende del otro, es decir, para poder calcular la distancia de las características, primero deberemos calcular dichas características, pero, tenemos la ventaja de que podemos ejecutar el primer paso y salir de la aplicación ya que, todos los procesos se guardan en ficheros en sus correspondientes carpetas. O sea que, si por ejemplo, calculamos las características de las imágenes y salimos de la aplicación, podremos obtener más tarde la distancia entre ellas sin la necesidad de volver al paso anterior.

En el paso número 3, podremos calcular la reducción de dimensionalidad con las dos opciones vistas (PCA y t-SNE). Se abrirá un sub-menú para que podamos elegir.

Ocurre lo mismo con el paso número 4, podremos elegir el método de clustering que queramos ver con el fin de probarlos todos y así poderlos comparar.

En el paso número 5, calcularemos primero los TP, TN, FP y FN, para ello, será necesario tener un Ground truth (en adelante, GT) de los datos, para poder obtener estos datos será necesario calcular las áreas de los rectángulos de cada cara que detecta el algoritmo empleado con el rectángulo del GT.

Para crear el GT, utilizaremos un proceso en *Matlab* el cual, para cada secuencia necesitaremos indicarle cuantas personas aparecen y donde se encuentran sus caras. Esto nos generará unos ficheros *Matlab* que más tarde tendremos que leer en *Python* y compararlos con los generados por los algoritmos empleados en la detección de caras.

Todo los pasos que hagamos en esta aplicación quedaran registrados mediante *logs*, estos *logs* se irán almacenando en una carpeta de históricos con el fin de rastrear posibles errores durante las ejecuciones.

La aplicación dispone de un sistema de carpetas que explicaremos a continuación. En primer lugar, necesitaremos modificar el archivo *tfg.ini* indicándole las rutas necesarias. Esto se ha pensado para poder introducir las imágenes sin necesidad de modificar el código y tener un nivel de abstracción del código mayor. Las carpetas necesarias que tendremos que introducir en el sistema serán las siguientes, aunque, con sólo introducir la carpeta *input*, el resto se irán creando automáticamente.

- *input*: Aquí tendremos que tener las imágenes que queremos analizar
- *output*: En este directorio, obtendremos las imágenes que aparecen caras, estas imágenes estarán sin tratar, simplemente, se habrá detectado alguna cara en ella.
- *boundingBox*: En este directorio, guardaremos las caras encontradas, solamente la cara, si en una imagen, encontramos más de una cara el nombre que encontraremos en este directorio será *i_nombre_de_la_imagen* , dónde i es el número de cara encontrado en la imagen original, yendo del 0 hasta el número de caras encontradas.
- *clusters*: En este directorio encontraremos las imágenes dentro de sub-directorios de los algoritmos utilizados, es decir, encontraremos por ejemplo un directorio llamado *agglomerative* y dentro las carpetas correspondientes con cada clúster.
- *features*: Aquí encontraremos cada característica guardada en un fichero, siguiendo el formato de nombre encontrado en el directorio *boundingBox*. También encontraremos un sub-directorio llamado *data* en el cual encontraremos las características de cada cara dentro de sus secuencias.
- *faces*: Aquí encontraremos las imágenes originales con todas las Bounding boxes encontradas.
- *mat*: Aquí podremos encontrar ficheros .mat (Matlab) que contendrán las Bounding boxes encontradas para cada imagen, esto nos servirá para poder calcular los *True positive, False positive, True negative, False negative* con las caras encontradas por el algoritmo de OpenFace.
- *matMatlab*: En este directorio, encontraremos los .mat, es igual que el directorio anterior, pero con la diferencia que aquí tendremos los Bounding boxes encontrados por el algoritmo de Matlab, FaceDetection.

5. Conclusiones y líneas futuras

En este proyecto hemos visto la metodología necesaria para la detección facial, la similitud entre caras, la reducción de dimensionalidad con la finalidad de mostrarlo y por último el agrupamiento.

En primer lugar, para la detección facial hemos visto dos métodos diferentes, uno con el lenguaje *Python* y otro para *Matlab*. Para ello, hemos tenido que aprender como funciona la detección facial y crear los ficheros necesarios para guardar las *Bounding boxes* encontradas.

Una vez encontradas las características de las caras con el algoritmo *OpenFace*, las hemos guardado en carpetas y también las características en ficheros.

Hemos recordado como calcular distancias entre caras y ver el umbral para poder decir que dos caras son la misma persona. Aunque podemos encontrarnos con imágenes que nos devuelva una distancia corta pero no se trate de la misma persona.

Un paso importante ha sido calcular el Ground Truth para cada secuencia y así saber cuantas personas tenemos por secuencia. Y gracias a esto, ha sido posible calcular para nuestro *dataset* los *True Positive, True Negative, False Positive y False Positive* y, después, gracias a estos calculos, sabemos que precisión tiene cada algoritmo.

Otra cuestión muy importante relacionado con el Ground Truth, era saber cuantas personas teníamos en nuestro *dataset*, no separado por secuencias, si no, cuantas personas tenemos en todo el *dataset*, hay personas repetidas en las secuencias y eso es lo realmente importante, concluimos con que tenemos 75 personas con el *dataset* que se han realizado las pruebas.

Y por último, tenemos la clasificación de las personas gracias a los pasos anteriores. Este paso, me ha servido personalmente para aprender mucho sobre el aprendizaje automático aunque, por desgracia, no he obtenido los resultados esperados con las pruebas realizadas. Me ha llevado mucho tiempo, pero me satisface saber que he aprendido y que esto no ha sido en balde, pues personalmente, quiero seguir investigando y aprendiendo sobre minería de datos.

Me gustaría que este proyecto haya podido ser un primer paso y que otro estudiante pudiese ampliarlo con nuevo conocimiento y fuese creciendo poco a poco.

Líneas futuras

En este apartado, vamos a explicar las posibles ideas que surgen después de realizar este proyecto. Ya que, como hemos mencionado en las conclusiones, personalmente, me gustaría que este proyecto creciera y este fuese un punto inicial.

Es cierto, que en este proyecto no hemos creado nada nuevo, pues hemos utilizado algoritmos ya existentes, pero, un punto inicial es darle forma juntando todos los algoritmos y explicando y entendiendo su funcionamiento en cada parte.

Una posible ampliación para este proyecto podría ser crear una aplicación web

con el fin de tener siempre disponible los pasos mencionados anteriormente, poder introducir imágenes y analizarlas como ya hemos visto. Podríamos ver los clústers creados, las características de cada imagen y escoger entre los algoritmos que queramos para poder ver los resultados. Para ello, podríamos crear una API, y que mediante las llamadas oportunas se pudiese consultar, y esto podría acabar en una aplicación para *smartphone*.

Otra posible ampliación de este proyecto podría ser, buscar otras alternativas al agrupamiento, es decir, intentar avanzar por redes neuronales y así poder tener la comparativa de que método es mejor.

Una posible ampliación más, podría ser, que mediante la cámara que portan los pacientes con deterioro cognitivo leve, fuese capaz de detectar la cara de la persona que está interactuando y obtener en tiempo real la clasificación y saber quien es la persona con la que interactua.

Si que es cierto que tenemos muchas oportunidades de expansión de este proyecto ya que, este campo de investigación está en auge y aún queda mucho camino por recorrer. Estas han sido las ideas principales.

A. Instalación del código

En este anexo vamos a proceder a explicar los pasos necesarios para poder ejecutar el código y así que sea más fácil poder probarlo. Para poder ejecutar este proyecto es indispensable tener instalado Python 2.7.x, lo podremos encontrar en *www.python.org*. La versión utilizada para el desarrollo del proyecto ha sido la 2.7.9. Con el fin de poder utilizar OpenFace necesitaremos instalar algun software extra, a continuación se muestran los pasos necesarios para su instalación.

Con tal de poder analizar las imágenes, será necesario tener instalado OpenCV, es código Open Source y muy potente para el tratamiento de imágenes disponible para Python. En la web de OpenCV podremos encontrar toda la documentación necesaria sobre su funcionalidad y también lo podremos descargar desde su portal (*www.opencv.org*)

También será necesario tener instalado dlib (*www.dlib.net*), este software es utilizado por OpenFace para los algoritmos de Machine learning y la extracción de las características.

Otro punto a instalar será el framework Torch (*www.torch.ch*) sirve para dar soporte a los algoritmos de aprendizaje en GPU.

Una vez tengamos todos los puntos anteriores instalados procederemos a instalar OpenFace. Podremos encontrar toda la información necesaria en la página principal (*https://cmusatyalab.github.io/openface/*).

Para la parte de Matlab, será necesario descargarlo de su web principal en el apartado de ventas, (*https://es.mathworks.com/*). Matlab no es OpenSource por lo que poder disponer de este software nos costará a partir de 35 Euros aproximadamente.

Este proyecto se ha desarrollado íntegramente desde Linux, con un procesador Intel Core i7-4790 3.60GHz. Y una tarjeta gráfica integrada Intel HD Graphics.

B. Coste aproximado del proyecto

En esta sección vamos a intentar hacer un desglose de lo que podría costarnos el proyecto o lo que podríamos cobrar por él. Es una aproximación ya que dependerá mucho de la experiencia del equipo de desarrollo, dado que, si se desarrolla por un equipo experimentado no harán falta tantas horas de búsqueda.

En primer lugar, antes de separar las fases que hemos seguido, es importante hacer la toma de decisiones de lo que necesitamos y queremos, en este caso el proyecto que queremos llevar a cabo.

Acto seguido, vamos a utilizar código open source, esto para calcular el coste va muy bien, porque abaratamos costes, por tanto, supongamos que solamente utilizamos OpenFace para la detección y la extracción de características y para ello necesitamos leer las imágenes que tenemos divididas por secuencia y modificar el código de OpenFace para compararlas y extraer la matriz de características. Como hemos mencionado anteriormente, este paso, puede ser muy rápido para alguien experimentado y conocedor de la materia, pero, en nuestro caso, no ha sido ha sido, aunque por otra parte, es la idea principal de hacer este proyecto, poder aprender y obtener conocimientos nuevos.

Por otra parte, necesitaremos, una vez obtenido las características ser capaces de montar la matriz, este paso es relativamente sencillo ya que no implica conocimientos en ningún algoritmo sino, implica conocimientos de programación.

Y, una vez tenemos esto, es hora de calcular el *Ground Truth*, este paso es sencillo, pero muy largo, es decir, no es un paso que necesitemos muchos conocimientos, pero si que es muy rutinario. Y por consiguiente, necesitaremos calcular el *Ground Truth* para todas las personas de nuestro dataset, este paso se puede extraer del anterior, lo que nos ahorrará tiempo, pero aún así será un paso largo.

Una vez tenemos los pasos anteriores, tendremos que clasificar las caras y separarlas por personas, este paso ha sido el más difícil pero, con gente experta en la materia se pueden obtener mejores resultados.

Por tanto, tenemos los diferentes puntos:

- Toma de decisiones
- Obtener las caras
- Crear la matriz de distancias
- Analizar y separar por caras
- Evaluación de los resultados

La intención de este anexo, no es poner un precio a todo el trabajo, ya que depende de muchos factores, como el equipo necesario, el tiempo disponible, la experiencia previa, etcétera. Pero, lo que si que queremos resaltar son los pasos necesarios para llevarlo a cabo y la evaluación de los puntos mencionados.

C. Envío de correo electrónico

En este apartado explicaremos el funcionamiento del uso del correo electrónico para conocer el estado de nuestro proyecto.

Con el fin de conocer el estado de la ejecución con indiferencia de los métodos utilizados y así conocer el estado de la ejecución, realizaremos envíos de correo electrónicos si, durante la ejecución ocurriera algún problema o, si por lo contrario, finalizará correctamente, enviaríamos un correo electrónico con el fichero de *logs* y así poder ver cómo ha ido y qué pasos se han ejecutado.

Para ello, crearemos una nueva cuenta de correo y así, gestionar el envío de correos electrónicos, con *HTML* creamos una vista insertándole el *logo* de la *Universidad de Barcelona* y un pequeño texto el cual, nos indicará el estado de la ejecución.

Envío correcto de e-mail

Figura 20: Resultado de la ejecución correcta

En la figura 20 podemos observar el ejemplo del correo electrónico que recibiríamos siempre y cuando la ejecución haya terminado y, no hayamos tenido ningún problema.

En cambio, en la figura 21 podemos observar el formato del correo electrónico que nos indica que ha habido problemas durante la ejecución. Tendremos que mirar el fichero adjunto para saber que ha ocurrido.

Error durante la ejecución

Figura 21: Error durante la ejecución

D. Web de presentación

En este anexo explicaremos la web que tenemos para explicar el proyecto de una forma dinámica y visual, explicando los algoritmos utilizados para ello, obtendremos un *template* y así obtener de forma sencilla toda la parte del diseño.

Dividiremos la web en diferentes partes, para poder explicar mejor cada paso necesario para el desarrollo de este proyecto. Y, podremos tener la web siempre accesible ya que, la tenemos publicada en *bitbucket*, juntamente con el código de la aplicación en la siguiente dirección *https://rubennieto.bitbucket.io/*.

Es una página estática, la cual modificaremos el código HTML y explicaremos el proyecto, de forma sencilla y resumida.

Extracto de la web

Figura 22: Breve explicación sobre los objetivos

En la figura 22 vemos una breve explicación que podremos encontrar en la web, sobre los objetivos de este proyecto.

En la figura 23 podemos observar una explicación sobre los pasos que hemos hecho para realizar el proyecto que se presenta.

Y por último, mostramos los algoritmos utilizados para la detección facial y el aprendizaje no supervisado, tal y como podemos ver en la figura 24.

Extracto de la web 2

Figura 23: Breve explicación la metodología

Extracto de la web 3

Figura 24: Algoritmos utilizados

E. Manual de usuario

En este apéndice explicaremos el uso de la aplicación desarrollada. Se trata de una aplicación desarrollada en Python la cual cuenta con un menú por consola y con él podremos obtener las características de las imágenes, calcular las distancias entre las caras obtenidas, entre otros.

Menú principal de la aplicación

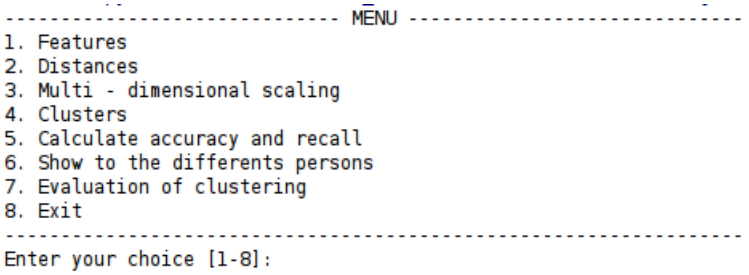


Figura 25: Menú de la aplicación

Como podemos ver en la imagen anterior (Figura 25), disponemos de un menú con ocho opciones. Estos pasos, por lo general, son secuenciales, es decir, necesitaremos tener el paso anterior para poder realizar el siguiente, esto está diseñado así debido a que, si queremos hacer pruebas podemos ejecutar diversos pasos cerrar la aplicación, para más tarde, si lo deseamos, continuar por donde nos habíamos quedado.

En la primera opción, podremos calcular las características de las imágenes. Es decir, introduciendo la ruta donde se encuentran las imágenes en el fichero de configuración *tfg.ini*, podremos obtener la detección facial y extraer las características de dichas imágenes.

En segundo lugar, podremos calcular las distancias entre las caras encontradas de nuestro *dataset*.

En tercer lugar, podremos hacer una reducción de la dimensionalidad con las características que hemos obtenido al ejecutar el paso 1. Este paso, nos llevará a un sub-menú para que podamos indicar el método que queramos utilizar. Los cuales pueden ser, PCA o t-SNE.

Reducción de dimensionalidad

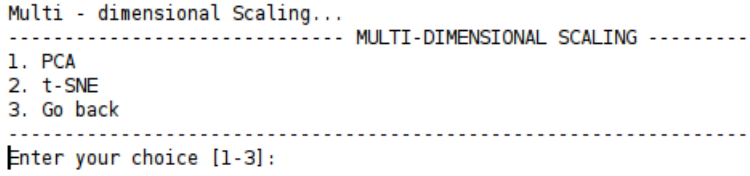


Figura 26: Menú para la reducción de dimensionalidad

En el cuarto elemento de nuestro menú encontramos con dos opciones, la primera

crear los clústers de forma no supervisada con diferentes métodos o por el contrario, crear una clasificación con aprendizaje supervisado.

En el siguiente punto del menú nos encontramos el cálculo del *accuracy* y *recall*. El cual, también obtendremos los True Positive, True Negative, False Positive y False Negative.

En el sexto punto del menú tenemos la interfície gráfica para visualizar las diferentes personas que tenemos en nuestro *dataset*.

Y por último, tenemos la opción de obtener la evaluación de los métodos utilizados, entre ellos se encuentra el clustering y la clasificación.

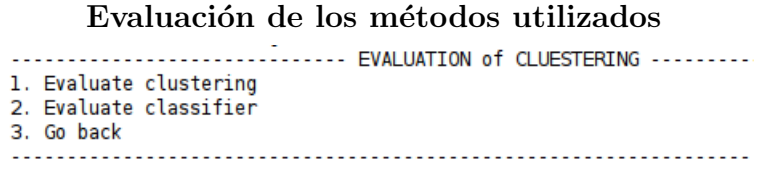


Figura 27: Menú para la evaluación de los métodos utilizados

F. Bag of Tracklets

En este apéndice hablaremos del proyecto de **Maedeh Aghaei**, el cuál a partir de un *dataset*, hace un seguimiento de la interacción del usuario que porta la cámara.

Este proyecto nos resulta interesante y por ello, lo utilizaremos para intentar obtener unos mejores resultados en nuestro trabajo.

Para ello, será necesario ejecutar el código de este proyecto sobre el *dataset* que disponemos y así obtener unos resultados que más tarde nos ayudaran a la clasificación. Cabe decir que, con este proyecto, obtenemos unos resultados verdaderamente muy buenos.

En primer lugar, leeremos los ficheros generados a partir del código, dichos ficheros tienen extensión *.mat*, es decir, ficheros de *Matlab* y como nuestro trabajo lo realizamos en *Python*, tendremos que leerlos y extraer los datos que en definitiva, son los diferentes *Bounding Boxes* de cada imagen de nuestra secuencia para cada persona.

En segundo lugar, nos guardaremos las caras encontradas siguiendo la estructura, es decir, para todas las personas que tenemos de cada secuencia guardaremos las en diferentes carpetas, así conseguimos tener una visualización más cómoda y sencilla.

Ejemplo de la estructura

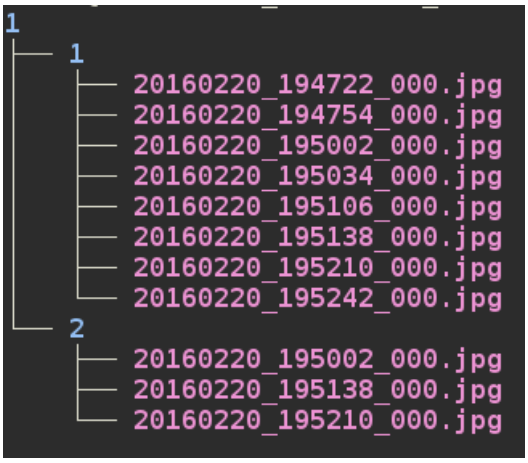


Figura 28: Bag of tracklet, ejemplo de la estructura obtenida

Como podemos ver en la figura 28, para la secuencia número uno, tenemos dos personas, en este caso le hemos llamado uno y dos y, contienen las diferentes imágenes donde aparecen en dicha secuencia.

Una vez tenemos los resultados, haremos clustering con los métodos que hemos visto con anterioridad y así observar los resultados obtenidos y calcular que método nos da unos mejores resultados.

Como podemos observar en la figura 29, el método que nos da unos mejores resultados es el método *Density-based spatial clustering of applications with noise* (DBSCAN). Aún así, nos da unos resultados parecidos a los obtenidos con *Agglomerative clustering* que hemos obtenido en las pruebas anteriores sin *Bag of Tracklet*.

Resultados del clustering con BoT

```
The result of evaluation of the AgglomerativeClustering is 0.0920
The result of evaluation of the Ward is 0.0959
The result of evaluation of the MeanShift is 0.1042
The result of evaluation of the Birch is 0.0916
The result of evaluation of the AffinityPropagation is 0.1019
The result of evaluation of the SpectralClustering is 0.0903
The result of evaluation of the DBSCAN is 0.2624
The result of evaluation of the MiniBatchKMeans is 0.0983
```

Figura 29: Clustering con los resultados de Bag of tracklet

Para poder realizar este agrupamiento calcularemos la media las características de las personas que hemos obtenido en cada secuencia y más tarde obtendremos la matriz de distancias para poder calcular los clústers.

Referencias

[1] Xiangxin Zhu and Deva Ramanan, Face Detection, Pose Estimation, and Landmark Localization in the Wild
https://www.ics.uci.edu/ xzhu/paper/face-cvpr12.pdf

[2] OpenFace, página principal,
https://cmusatyalab.github.io/openface/

[3] Algoritmo K-Means,
http://jarroba.com/k-means-python-scikit-learn-ejemplos/.
https://cmusatyalab.github.io/openface/.

[4] Python, página principal,
https://www.python.org.

[5] OpenCV, página principal,
www.opencv.org.

[6] Dlib,
www.dlib.net.

[7] Torch,
www.torch.ch.

[8] Matlab,
https://es.mathworks.com/.

[9] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, Cordelia Schmid, Deep-Matching
https://lear.inrialpes.fr/src/deepmatching/deepmatching_submitted_ijcv.pdf

[10] By Walber - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=36926283.

[11] Precision, recall, f-measure,
http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html.

[12] Overlap between rectangles
http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[13] Maedeh Aghaeia, Mariella Dimiccolia, Petia Radeva, Multi-Face Tracking by Extended Bag-of-Tracklets in Egocentric Videos
https://arxiv.org/pdf/1507.04576v2.pdf

[14] Extracción de metadatos
https://smarnach.github.io/pyexiftool/

[15] Métodos Jerárquicos de Análisis
http://www.ugr.es/~gallardo/pdf/cluster-3.pdf

[16] Spectral Clustering
https://charlesmartin14.wordpress.com/2012/10/09/
spectral-clustering/

[17] Unsupervised Learning
http://scikit-learn.org/stable/unsupervised_learning.html\
#unsupervised-learning/

[18] Ejemplo de k-Means
https://dlegorreta.wordpress.com/tag/k-means-clustering/

[19] Shaogang Gong, Marco Cristani, Shuicheng Yan, Chen Change Loy. Person Re-Identification
http://link.springer.com/book/10.1007/978-1-4471-6296-4

[20] Zhanpeng Zhang, Ping Luo, Chen Change Loy, Xiaoou Tang. Joint Face Representation Adaptation and Clustering in Videos
https://link.springer.com/chapter/10.1007/978-3-319-46487-9_15

[21] Clústers Jerárquicos
https://es.wikipedia.org/wiki/Agrupamiento_jer\%C3\%A1rquico

[22] Clasificación y regresión
https://inteligenciaartificial101.wordpress.com/2015/10/20/
aprendizaje-supervisado/

[23] Analisis de cluster y árboles de clasificación
http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/
tema6dm.pdf

[24] TFG sobre detección facial
http://dugi-doc.udg.edu/bitstream/handle/10256/1953/2\%20PFC\
%20Sureda\%20Casta\%C3\%B1al.pdf?sequence=2

[25] Artículo sobre OpenFace y machine learning
https://medium.com/\@ageitgey/machine-learning-is-fun-part-4-modern-face\
-recognition-with-deep-learning-c3cffc121d78\#.yedax5db1