DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP2 - Cara a cara

Autovalores, autovectores, método de la potencia, SVD

Métodos Numéricos

Primer Cuatrimestre de 2017

Integrante	LU	Correo electrónico
Len, Julián	467/14	julianlen@gmail.com
Len, Nicolás	819/11	nicolaslen@gmail.com
Mascitti, Augusto	954/11	mascittija@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Índice

1. Resumen

En el siguiente trabajo práctico, analizaremos una de las aplicaciones de Autovalores y Autovectores.

En este caso, haremos un pequeño sistema de reconocimiento de caras. Para esto se usarán conceptos de **análisis de componentes principales** (**PCA**). Este concepto, nos servirá para crear una **transformación característica** (**TC**), la cual operará con los autovectores de la matriz de entrenamiento, tomando las componentes principales necesarias y generando un cambio de base. Es decir, busca reducir la cantidad de dimensiones de la información con la que se cuenta, y simultáneamente, busca que las nuevas variables tengan información representativa para clasificar los objetos de la base de entrada, o sea, a la hora de reconocer a qué cara pertenece una nueva imagen, aplicarle el mismo cambio de base, y contar con la información suficiente para determinar a qué persona pertenece. Finalmente, se experimentará sobre esto último, o sea, sobre la fase del programa que toma la decisión de a quién corresponde la cara de la imagen nueva.

Para esto, como instancias de entrenamiento, se tiene un conjunto de N personas, cada una de ellas con M imágenes distintas de sus rostros en escala de grises del mismo tamaño y resolución. Cada una de estas imágenes, están correctamente etiquetadas y sabemos a qué persona corresponde.

2. Introducción

2.1. Objetivos Generales

En este trabajo vamos a estudiar distintos métodos de aprendizaje automático para solucionar el problema de reconocimiento de caras.

Para abordar el tema se plantea desarrollar un clasificador, que a partir de una base de datos de caras identificadas (es decir, en la que ya sabemos a qué persona pertenece cada cara), pueda agruparlas según determinadas características.
Esta etapa de agrupamiento es la parte de entrenamiento que realiza el algoritmo.

Luego trabajaremos con caras nuevas, para las que el clasificador utilizará la agrupación obtenida anteriormente para identificar a qué persona corresponde cada una.

La idea general detrás de los clasificadores que haremos es tomar a cada imagen, como un punto ubicado en un espacio multidimensional, para luego medir la distancia contra una nueva imagen y poder deducir a quién pertenece dicha imagen.
Es decir que cada imagen se puede interpretar como un punto en un espacio de n dimensiones. Pudiendo ser n la cantidad de píxeles totales de la imagen, o como veremos más adelante, con un n mucho menor.
Luego, para obtener una predicción de a qué persona puede pertenecer una cara no etiquetada, buscamos cuál de las que sí tienen clasificación, está más cerca de ésta nueva en el espacio de n dimensiones antes mencionado, y le asignamos esa etiqueta a la nueva.

Esta explicación no es más que una simplificación para introducir el tema; veremos en las secciones siguientes el detalle del algoritmo y las variaciones que consideraremos.

Para el entrenamiento y las pruebas de nuestro algoritmo trabajaremos con la base ORL faces, compuesta por 41 sujetos con 10 imágenes por sujeto de sus respectivas caras.

Antes de meternos en el detalle de los métodos implementados repasemos algunos conceptos teóricos que servirán más adelante.

2.2. Autovalores y autovectores

Los **autovalores** asociados a una transformación lineal son aquellos vectores que conservan su dirección al aplicarse la transformación lineal, es decir, al ser multiplicados por la matriz asociada a la transformación. Los **autovalores**, en este caso, indican el coeficiente en el que se reduce o se amplia el tamaño del autovector.
Para calcularlos es necesario despejar el siguiente sistema de ecuaciones:

$$\lambda \in \mathbb{C} \text{ es autovalor de } A \in \mathbb{R}^{nxn}, \text{ con autovector asociado } v \text{ si y solo si } Ax = \lambda x$$

Una forma de calcular los *autovalores* es calcular las raices del **polinomio característico** $P(\lambda)$. El cual se obtiene haciendo el siguiente cálculo:

$$P(\lambda) = det(A - \lambda I)$$

Donde $P(\lambda)$ es el *polinomio característico* de grado n y sus raices son los *autovalores*. Luego podemos calcular los *autovectores* asociados a cada *autovalor* resolviendo el sistema $(\mathrm{A} - \lambda I)v = 0$, reemplazando λ por cada uno de los *autovalores* encontrados. El resultado luego de resolver el sistema es el conjunto de generadores de los *autovectores* asociados al *autovalor* usado. Puede ser uno o más en caso de ser el *autovalor* una raíz multiple del polinomio característico.
El autovector principal o dominante de una transformación lineal es el *autovector* asociado al *autovalor* más grande.

2.3. Método de la potencia y deflación

Para el cálculo de los autovalores y autovectores de la matriz vamos a utilizar el **método de la potencia** reiteradas veces, con **deflación** en cada paso.

2.3.1. Método de la Potencia

Este método nos permite realizar una aproximación para el cálculo del polinomio característico de una matriz. Para aplicarlo pedimos como hipótesis, que $A \in \mathbb{R}^{nxn}$ tenga n autovalores tales que $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ con una base de autovectores asociados $v_1, v_2, \ldots, v_n$. Es importante que exista el *autovalor dominante* λ_1 y que el valor inicial x_0 no sea ortogonal al autovector asociado a λ_1.

Sea $A^{n\times n}$, $\lambda_1, ..., \lambda_n$ autovalores v_1, $v_2, ..., v_n$ autovectores linealmente independientes. Supongamos que $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$ donde λ_1 es el autovalor dominante.

Como $v_1, ..., v_n$ son una base, entonces $\exists \beta_j$ tal que:

$$x = \sum_{j=1}^{n} \beta_j v_j \forall x \in \Re^n$$

$$Ax = \sum_{j=1}^{n} \beta_j A v_j = \sum_{j=1}^{n} \beta_j \lambda_j v_j$$

$$\Rightarrow A^2 x = \sum_{j=1}^{n} \beta_j \lambda_j^2 v_j = x^{(1)}$$

Defino la sucesión

$$x^{(k)} = A^k x = \sum_{j=1}^{n} \beta_j \lambda_j^2 v_j = x^{(1)} = \lambda_1^k \sum_{j=1}^{n} \beta_j (\frac{\lambda_j}{\lambda_1}^k v_j)$$

Como vale que $\lambda_j/\lambda_1 < 1$ entonces

$$\lim_{k\to\infty} A^k x = \lim_{k\to\infty} \lambda_1^k \beta_1 v_1$$

Por lo tanto tenemos que $x^{(k)} = \lambda_1^k (v^{(1)}\beta_1 + \epsilon^{(k)})$ con $\epsilon^{(k)}_{k\to\infty} \to 0$. Entonces sea ϕ una función lineal tal que:

$$\phi(v_1, \beta_1) \neq 0 \Rightarrow \phi(0) = 0$$

Entonces

$$\frac{\phi(x^{k+1})}{\phi(x^k)} = \lambda_1 (\frac{\phi(b^{(1)}\beta_1) + \phi(\epsilon^{k+1})}{\phi(b^{(1)}\beta_1) + \phi(\epsilon^k)})$$

$$\Rightarrow \lim_{k\to\infty} \frac{\phi(x^{k+1})}{\phi(x^k)} = \lambda_1$$

Una función lineal que se podría utilizar puede ser tomar $\phi(x) = x_i$ para algún i.

2.3.2. Deflación

El método de las potencias nos permite calcular el autovector asociado al autovalor dominante de una matriz. Vayamos un paso más adelante, y veamos cómo obtener los autovectores asociados a los α autovalores más grandes.

Ahora no alcanza con aplicar el método de las potencias una sola vez a las matrices con las que trabajemos, sino que vamos a necesitar hacerlo tantas veces como autovectores busquemos. Para esto será necesario un último paso: la **deflación**.

La *deflación* se basa en el siguiente teorema: Sea A una matriz con $\lambda_1, \ldots, \lambda_n$ autovalores distintos y una base ortogonal de autovectores. Entonces la matriz $B = A - \lambda_1 v_1 v_1^t$ son $0, \lambda_2, ..., \lambda_n$, siendo estos autovalores de A también, con autovectores asociados $v_1, ..., v_n$.

Es decir que, una vez obtenido mediante el método de las potencias el autovalor dominante y su autovector asociado, podemos aplicar la operación $B = A - \lambda_1 v_1 v_1^t$. Luego buscamos el autovalor dominante de B, que a su vez es autovalor de la matriz A original.
Realizando el mismo proceso α veces se obtienen los autovectores que necesitamos.

2.4. Descomposición SVD

2.4.1. SVD

Sea $A \in \mathbb{R}^{mxn}$ una matriz de rango r y $A = U\Sigma V^t$ su descomposición en valores singulares (SVD), con $U \in \mathbb{R}^{mxm}$, $\Sigma \in \mathbb{R}^{mxn}$ y $V \in \mathbb{R}^{nxn}$, siendo $\Sigma = diag\{\sigma_1, \sigma_2, \ldots, \sigma_r, 0, \ldots, 0\}$ y $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. Llamamos a σ_i el i-ésimo valor singular. Sean $v_1, \ldots, v_n$ las columnas de V y $u_1, \ldots, u_m$ las columnas de U,

1. $v_1, \ldots, v_n$ son autovectores de A^tA,
2. $u_1, \ldots, u_m$ son autovectores de AA^t,
3. $\lambda_i = \sigma_i^2$ son los autovalores de A^tA asociados al autovector v_i,
4. $$Av_i = \begin{cases} \sigma_i u_i & \text{i} \leq r \\ 0 & cc \end{cases}$$
5. $$A^t u_i = \begin{cases} \sigma_i v_i & \text{i} \leq r \\ 0 & cc \end{cases}$$

2.4.2. Relación entre $M = X^tX$ y $\overline{M} = XX^t$

Si se obtienen los autovalores y autovectores de $\overline{M}$, podremos obtener los autovalores y autovectores de M haciendo uso de las propiedades de la descomposición SVD.

Sea $A^t = X$, sea $A = X^t$, sea $\overline{M} = XX^t = A^tA$, sea $M = X^tX = AA^t$ y sea $A = U\Sigma V^t$. Las raíces cuadradas de los autovalores de $\overline{M}$ serán la diagonal de Σ (y por lo tanto, también serán los autovalores de M) y sus autovectores asociados serán las columnas de V.

$$\Sigma =_{(2.)} \begin{bmatrix} \sigma_1 & & & & & & \\ & \sigma_2 & & & & & \\ & & \ddots & & & & \\ & & & \sigma_r & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \end{bmatrix} V =_{(1.)} \begin{bmatrix} v_1 & & & \\ & v_2 & & \\ & & \ddots & \\ & & & v_r \end{bmatrix}$$

Por lo tanto, al despejar U de $A = U\Sigma V^t$ obtendremos los autovectores de $M = X^tX = AA^t$. Es decir,

$$U = \begin{bmatrix} u_1 & u_2 & \ldots & u_r \end{bmatrix} =_{(4.)} \begin{bmatrix} \frac{Av_1}{\sigma_1} & \frac{Av_2}{\sigma_2} & \ldots & \frac{Av_r}{\sigma_r} \end{bmatrix}$$

En este Trabajo Práctico, la herramienta que tenemos para calcular autovalores y autovectores es el método de la potencia que tiene un rol importante en la performance del programa.

A la hora de calcular los autovalores y autovectores, en vez de aplicar el método de la potencia a M, tenemos como alternativa aplicar el método de la potencia a $\overline{M}$ y con las propiedades de la descomposición SVD, calcular los autovalores y autovectores de M. Dado el costo de las operaciones, conviene usar esta alternativa cuando M es una matriz mucho más grande que $\overline{M}$.

2.5. Descomposición en Componentes Principales (PCA)

PCA, por sus siglas en inglés Principal Component Analysis (análisis de componentes principales) es un método para reducir la dimensionalidad de los vectores.

Recordemos que si pensamos a la imagen como un punto en un espacio de dimensión cantidad de píxeles totales de la imagen, a la hora de realizar cálculos con imágenes relativamente grandes, manipular esa cantidad enorme de datos conlleva una pérdida de performance importante.

La intuición de este método es que dado un set de n muestras con m variables (para algún m mayor a 1), éste puede ser transformado en otro con menor cantidad de variables que no serán las mismas del set original, sino que representarán las características que más difieren dentro de éste y serán lo más independientes entre sí posible.

Antes de analizar el método, hay que ver qué significa que las nuevas componentes sean independientes. Con PCA, ésto se define en términos de la varianza de los datos originales: la dependencia entre variables se elimina buscando las direcciones en las que hay más varianza y son estas direcciones las que definen la nueva base.

Veamos a continuación el detalle del método.

Para $i = 1, \ldots, n$, $x_i \in \mathbb{R}^m$ corresponde a la i-ésima imagen de nuestra base de datos almacenada por filas en un vector, y sea $\mu = (x_1 + \cdots + x_n)/n$ el promedio de las imágenes. Definimos $X \in \mathbb{R}^{nxm}$ como la matriz que contiene en la i-ésima fila al vector $(x_i - \mu)^t/\sqrt{n-1}$, y $M = X^t * X$ a la matriz de covarianza de la muestra X.

Recordemos que la covarianza se puede pensar como una medida de cuán correlacionadas están dos variables (cuando las dos variables son la misma hablamos de varianza).

Como M es una matriz cuadrada podemos calcularle los autovalores y autovectores (para lo cual usaremos el método de la potencia, explicado más adelante), normalizando estos últimos ya que PCA requiere tenerlos de esta manera.

El autovector asociado al autovalor de mayor valor corresponde a la dirección de mayor varianza. El autovector asociado al siguiente autovalor es el que le sigue y así siguiendo con cada uno de mayor a menor.

Siendo v_j el autovector de M asociado al j-ésimo autovalor λ_j al ser ordenados por su valor absoluto, definimos para $i = 1, \ldots, n$ la transformación característica de la cara x_i como el vector $\mathbf{tc}(x_i) = (v_1^t x_i, v_2^t x_i, \ldots, v_\alpha^t x_i) \in \mathbb{R}^\alpha$, donde $\alpha \in \{1, \ldots, m\}$ es un parámetro de la implementación.

Este proceso corresponde a extraer las α primeras componentes principales de cada imagen. La intención es que $\mathbf{tc}(x_i)$ resuma la información más relevante de la imagen, descartando los detalles o las zonas que no aportan rasgos distintivos.

Dada una nueva imagen x de una cara, que no se encuentra en el conjunto inicial de imágenes de entrenamiento, el problema de reconocimiento consiste en determinar a qué persona de la base de datos corresponde. Para esto, se calcula $\mathbf{tc}(x)$ y se compara con $\mathbf{tc}(x_i)$, para $i = 1, \ldots, n$, utilizando un criterio de selección adecuado.

2.6. Algoritmos de Clasificación

Como ya se mencionó anteriormente, el trabajo consiste en un algoritmo que aprenda y pueda luego clasificar imágenes.

Esto se divide en dos etapas: en la primera nos concentraremos sólo en el entrenamiento del clasificador, deshaciéndonos de la información redundante y cuyo aporte sea menos relevante de forma que podamos trabajar con menos información que la inicial, pero más relevante.

Luego, con imágenes que no fueron tomadas en cuenta a la hora de entrenar al clasificador, someter a distintos algoritmos de clasificación a la prueba de reconocimiento de a quién pertenece una cara nueva.
Acá vamos a realizar distintos test para verificar qué tan preciso resulta ser el clasificador.

Se trata de algoritmos basados en la idea previamente introducida de buscarle vecinos cercanos al dato que queremos clasificar.
Así, cada vez que el algoritmo recibe una imagen x sin clasificar, busca entre todas las imágenes, a las más cercanas, o sea, busca a las que minimizan $\|x_i - x\|$
Se trata de un algoritmo muy simple e intuitivo, pero presenta dos desventajas a considerar:

- **Dimensionalidad**: éste método sufre de un fenómeno conocido como ”maldición de la dimensionalidad”. Lo que sucede al trabajar en dimensiones tan grandes es que los puntos quedan muy alejados unos de otros, diluyendo así el concepto de vecindad.
- **Base de entrenamiento sesgada**: Para el caso particular de kNN, sucede al trabajar con una base de entrenamiento con muchos elementos de una categoría (digamos, A) y muy pocos de otra (digamos, B). A la hora de clasificar un elemento nuevo y dependiendo del k elegido, puede pasar que muchos elementos de A entren en la selección de vecinos, aumentando la probabilidad de categorizar al nuevo elemento como A.

El primer problema lo atacamos reduciendo las dimensiones con PCA, generando transformaciones carácteristicas que nos lleven a las imágenes de una representación de n dimensiones (siendo n la de cantidad total de píxeles), a α dimensiones.

Y el segundo, lo dejaremos por un momento de lado, asumiendo que no va a ser un conflicto en nuestro desarrollo, ya que la base de entrenamiento la realizamos con la misma cantidad de imágenes por persona. Pero es posible que en la etapa de experimentación podamos concluir algo en relación a esto.

2.6.1. K vecinos más cercanos (kNN)

Comenzamos con la idea de quedarnos con aquel vecino que minimice su distancia al de la nueva imagen. Se debe tomar en cuenta que este algoritmo, es muy susceptible a la dispersión de los datos.

Puede pasar que no todas las imágenes de una misma persona se acumulen alrededor de un punto en el espacio. Es factible que alguna tenga un desvío estándar un poco mayor que el resto de las imágenes.

De esta forma, pensamos que puede suceder tranquilamente que una imagen termine teniendo una transformacion carácteristica que la ubique en el espacio muy cerca de una que no está dentro de su clase de equivalencias, más allá de encontrarse relativamente cerca del resto de las de su clase.
Como una propuesta superadora, nos dimos cuenta que podíamos tomar k vecinos más cercanos, y entre ellos decidir a cuál pertenece la nueva imagen. Por ejemplo, si una nueva imagen de la persona 3, tiene como vecino más cercano a una imagen de la persona 5, pero las próximas 5 imágenes más cercanas pertenecen a la persona 3, podemos asumir que clasificar a la nueva imagen como perteneciente a la persona 3, sería lo más correcto.

Luego nos dimos cuenta que únicamente contar repetidos, puede generar confusión a la hora de registrar un empate entre dos clases distintas. Con lo cual para mitigar este problema decidimos darle un peso al orden de la distancia de la nueva imagen, a las k imágenes más cercanas.

De esta manera, realizando un análisis ponderado de las distancias, se termina eligiendo un candidato a clasificación.

$f(x) =$ persona a la que pertenece la imagen x

$$w_i = \frac{1}{\|x - x_1\|^2}$$

$$\delta(a,b) = \begin{cases} 1 & si \quad a = b \\ \\ 0 & si \quad a \neq b \end{cases}$$

$$f^{'}(x) = argmax_{p \in Personas} \sum_{i=1}^{k} w_i \delta(p, f(x_i))$$

2.6.2. Centro de masa más cercano

En el segundo método de elección, calculamos por cada persona su centro de masa. Es decir, sea V_i^j la j-ésima transformación característica de la imágen de la i-ésima persona. Y sea *K*, la cantidad de imágenes por persona para la base de entrenamiento. Calculamos su centro de masa (*C*) como,

$$C = \sum_{i=1}^{K} V_i^j / K$$

De esta manera, obtenemos, por persona, un punto en el espacio. Luego, por cada imagen entrante calculamos su transformación característica, para hallar a que punto del espacio se acerca más. Encontrando de esta forma, el Centro de Masa más cercano a la nueva imágen, teniendo así a qué persona pertenece.

2.7. Métricas

La calidad de los resultados obtenidos sera analizada mediante diferentes métricas. En particular, la métrica más importante que debe reportarse en los experimentos, es la tasa de efectividad lograda, es decir, la cantidad de caras correctamente clasificadas respecto a la cantidad total de casos analizados.

2.7.1. HitRate

El Hit Rate, es la medida que toma la cantidad de aciertos de un clasificador, sobre la cantidad total de intentos. Es decir, si para el clasificador C, se tiene que la clase i, logró Vp aciertos luego de que hayan n pruebas. Entonces el hit rate es $\frac{Vp}{n}$

2.7.2. Precisión

Es una medida de cuántos aciertos relativos tiene un clasificador dentro de una clase particular. Es decir, dada una clase i, la precisión de dicha clase es $\frac{tp_i}{tp_i + fp_i}$

En la anterior fórmula, tp_i son los *verdaderos positivos* de la clase i. Es decir, muestras que realmente pertenecían a la clase i y fueron exitosamente identificadas como tales.

En contraposición, fp_i son los *falsos positivos* de la clase i. Son aquellas muestras que fueron identificadas como pertenecientes a la clase i cuando realmente no lo eran.

Luego, la precision en el caso de un clasificador de muchas clases, se define como el promedio de las precision para cada una de las clases.

2.7.3. Recall

Es una medida de qué tan bueno es un clasificador para, dada una clase particular, identificar correctamente a los pertenecientes a esa clase. Es decir, dada una clase i, el recall de dicha clase es $\frac{tp_i}{tp_i+fn_i}$

En la anterior fórmula, fn_i son los *falsos negativos* de la clase i. Es decir, muestras que pertenecían a la clase i, pero que fueron identificadas con otra clase.

Luego, el recall en el caso de un clasificador de muchas clases, se define como el promedio del recall para cada una de las clases.

3. Desarrollo

3.1. Implementación

En esta parte del trabajo práctico, pensamos cuál era la mejor opción para desarrollar los métodos deseados, y poder analizarlos y compararlos.
El programa recibe dos parámetros:

- La ruta de entrada del archivo que contendrá información acerca de las imágenes con las que se entrenará a la base de datos, agrupadas por persona.
- La ruta de salida donde se guardarán los autovalores de la matriz de covarianza ordenados de mayor a menor.

Para empezar, decidimos utilizar la librería Eigen[1] que nos facilita la creación, edición y las operaciones elementales entre matrices y vectores.
Luego implementamos el algoritmo necesario para leer una imagen de tipo PGM (Portable GreyMap) y obtener sus datos (ancho, alto) y su matriz donde cada valor es un entero entre 0 y 255 que representa un color dentro de la escala de grises, siendo 0 el color negro y 255 el color blanco. A partir de estos valores, se crea la matriz con Eigen para luego poder manipularla fácilmente.

El programa comienza entrenando a la base de datos, y esto consta de varios pasos:

3.1.1. Imágenes y esperanza

- Imágenes

  Se crea una matriz $A \in \mathbb{R}^{nxm}$. Siendo n la cantidad de imágenes total (es decir, la cantidad de personas por la cantidad de imágenes por persona), y m la cantidad de píxeles (ancho x alto) de cada imagen (vale aclarar que todas las imágenes tendrán el mismo tamaño). Esta matriz contendrá en la fila i, el vector que representa a la matriz aplanada de la imagen i $(0 \leqslant i < n)$

$$A = \begin{bmatrix} imagen_1 \\ imagen_2 \\ \cdots \\ imagen_n \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ \cdots \\ A_n \end{bmatrix}$$

- Esperanza

  Se crea un vector μ que contendrá la suma de los vectores (matrices aplanadas) de todas las imágenes, sobre la cantidad total de imágenes. Este vector contendrá la media de las imágenes.

$$\mu = \left[\frac{A_1 + A_2 + \cdots + A_n}{n}\right]$$

3.1.2. Matriz de covarianza

A partir de la matriz

$$X = \begin{bmatrix} \frac{(A_1-\mu)^t}{\sqrt{n-1}} \\ \frac{(A_2-\mu)^t}{\sqrt{n-1}} \\ \cdots \\ \frac{(A_n-\mu)^t}{\sqrt{n-1}} \end{bmatrix}$$

Se obtiene la matriz de covarianza $M = X^t X$ para luego obtener sus autovalores y autovectores.

Dependiendo de la dimensión de X, se puede tomar la decisión de obtener los autovalores y autovectores de M, a partir de $\overline{M} = XX^t$, sus autovalores y autovectores asociados.
Es decir, como $X \in \mathbb{R}^{nxm}$, entonces $M \in \mathbb{R}^{mxm}$ y $\overline{M} \in \mathbb{R}^{nxn}$. Por lo tanto, si $m < n$, será menos costoso realizar el método de la potencia con M. Y si $n < m$, será menos costoso utilizar $\overline{M}$ para iterar

en el metodo e la potencia. Esta decisión tiene grandes repercusiones en la complejidad tanto espacial como temporal del algoritmo.

3.1.3. Entrenamiento

Dada la cantidad de componentes principales que se desean obtener (llamamos k a la cantidad de componentes principales), dependiendo de la decisión tomada en el paso anterior, el entrenamiento varía. En ambos casos, se hace uso del método de potencia con deflación k veces para calcular los k autovalores y sus autovectores asociados, ya sea de M o de $\overline{M}$.

Vale aclarar que el método de la potencia retorna el autovalor que tenga el mayor valor absoluto. Por lo tanto, al correrlo k veces, se obtendrán los k autovalores (y sus autovectores) ordenados de mayor a menor por su valor absoluto.

Algorithm 1 Método de la potencia

```
1: function METODOPOTENCIA(B, x_0, niter)
2:     v ← x_0
3:     for i ∈ 1..niter do
4:         v ← Bv / ||Bv||
5:         λ_i ← (v^t B v) / (v^t v)
6:         if |λ_i − λ_{i−1}| < ε then
7:             return λ, v
8:         end if
9:     end for
10:    return λ, v
11: end function
12:
13: function DEFLACION(B, λ, v)
14:    return B − (λ ∗ v ∗ v^t)
15: end function
```

En caso de que se trabaje con la matriz M, ya se tendrán los k autovalores y autovectores asociados, por lo que no será necesario hacer nada más.

En cambio, en caso de que se trabaje con la matriz $\overline{M}$, se hará uso de las propiedades de la factorización SVD para calcular los k autovalores y autovectores asociados de M a partir de los de $\overline{M}$.

Es decir, sea $0 \leqslant i < k$, por cada autovalor λ_i y su autovector asociado v_i de $\overline{M}$ que se obtenga, λ_i también será el i-ésimo autovalor de M y su autovector asociado se calculará de la siguiente manera $w_i = \frac{X^t * v_i}{\sqrt{\lambda_i}}$.

En ambos casos, el entrenamiento retornará una tupla (λ, v) que contendrá como primer miembro un vector de autovalores (ordenados de mayor a menor por su valor absoluto), y como segundo miembro una matriz donde cada columna será un autovector.

$$\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \dots \\ \lambda_k \end{bmatrix} \quad v = \begin{bmatrix} v_1 & v_2 & \dots & v_k \end{bmatrix}$$

3.1.4. Transformación característica

A partir de la matriz de imágenes A, y sea v_j $(0 \leqslant j < k)$ el autovector de M asociado al j-ésimo autovalor ordenados por su valor absoluto, se define la *Transformación Característica* de la cara A_i como el vector $tc(A_i) = (v_1^t A_i, v_2^t A_i, \dots, v_k^t A_i)$.

Dada la definición de una transformación característica, para mejorar la performance del algoritmo, tomamos la decisión de calcular las **tc** de todas las caras en una única operación de la siguiente manera:

$$tc(A) = \begin{bmatrix} A_1 \\ A_2 \\ \dots \\ A_n \end{bmatrix} \begin{bmatrix} v_1 & v_2 & \dots & v_k \end{bmatrix} = \begin{bmatrix} A_1v_1 & A_1v_2 & \dots & A_1v_k \\ A_2v_1 & A_2v_2 & \dots & A_2v_k \\ \dots & & & \\ A_nv_1 & A_nv_2 & \dots & A_nv_k \end{bmatrix} = \begin{bmatrix} v_1^tA_1 & v_2^tA_1 & \dots & v_k^tA_1 \\ v_1^tA_2 & v_2^tA_2 & \dots & v_k^tA_2 \\ \dots & & & \\ v_1^tA_n & v_2^tA_n & \dots & v_k^tA_n \end{bmatrix}$$

Este proceso corresponde a extraer las *k* primeras componentes principales de cada imagen. La intención es que $tc(A_i)$ resuma la información más relevante de la imagen, descartando los detalles o las zonas que no aportan rasgos distintivos.

A esta matriz $tc(A)$ la llamamos matriz de coordenadas. Y tiene las coordenadas del cambio de base de nuestras imagenes de entrenamiento a un espacio de α dimensiones.

Dado que ya tenemos los autovalores de M, imprimimos en un archivo el vector solución con las k raíces cuadradas de los autovalores de mayor magnitud, con un valor por línea.

3.1.5. Clasificación de caras

A partir de las rutas que contiene el input de entrada, se obtienen las imágenes con caras para detectar a qué persona pertenece cada una.

- **Transformación característica de la nueva imagen**

  Para cada vector (matriz aplanada) w_j que contiene una de estas imágenes, se calcula **tc(**w_j**)** a partir de la matriz que contiene los autovectores de M (la llamamos V, donde cada columna es un autovector). Es decir,

  $$tc(w_j) = w_j^t \begin{bmatrix} v_1 & v_2 & \dots & v_k \end{bmatrix}$$

  Esto nos devuelve como resultado un vector con las α coordenadas de nuestra nueva imagen. las cuales vamos a utilizar para calcular la distancia contra el resto de las imágenes.

- **K vecinos más cercanos**

  Una vez obtenida la $tc(w_j)$, se necesita decidir a qué persona pertenece la imagen w_j.

  Para esto, la función *kVecinosMasCercanos* toma como parámetros la cantidad de vecinos de la matriz de entrenamiento transformada, más cercanos a $tc(w_j)$. Según el k que sea pasado como entrada, por motivos de complejidad, la manera de encontrar el vecino más cercano varía:

  - **Para k** $= 1$: Para esto se recorrerá todas las imágenes de la base de entrenamiento (con la **TC** aplicada), y se calculará la mínima distancia entre éstas y $tc(w_j)$. Finalmente, se devolverá a qué persona pertenece w_j, resolviéndolo en tiempo lineal.
  - **Para k** > 1: En este caso se almacenan las distancias de nuestra imagen nueva, contra todas las imágenes que hay en la base de entrenamiento. Luego se ordena de menor a mayor por dicha distancia. Y por último se realiza una votación ponderada entre los primero k vecinos más cercanos. Retornando aquel que sale electo en esta votación.

    La complejidad se ve modificada con respecto al algoritmo con $K = 1$ en la necesidad de ordenar los datos y luego calcular la votación.

- **Centro de masa más cercano**

  Para calcular los centros de masa, dado **M** la matriz de entrenamiento (con la **TC** aplicada), con **K** la cantidad de imágenes por persona, se genera una nueva matríz $M^{(2)}$ de dimensión *cantPersonas x Dimensiones* tal que en la fila iésima de $M^{(2)}$

  $$M_i^{(2)} = \sum_{C}^{C+K} M_j/K$$

Siendo **C**, la fila donde empiezan las imágenes de la iésima persona.

Luego de obtener $M^{(2)}$, la matriz de centros de masa, se obtiene un nuevo vector de una imágen nueva, con su **TC**, y se busca la distancia mínima entre las filas de **M**, obteniendo así a que centro de masa (o persona) pertenece.

4. Experimentación

4.1. Metodología

Para poder experimentar y ajustar los parámetros de nuestros métodos de reducción de dimensiones adecuadamente, debemos poder saber que tan bien filtran la información poco relevante. Al contar con una base de información debidamente etiquetada, podemos utilizarla como parámetro de comparación.

La idea es tomar una porción de la misma para entrenar el método y los datos restantes como evaluación. Como contamos con todas las instancias de la misma debidamente etiquetada, podemos saber realmente, dada una métrica de calidad elegida, qué tan bien se desempeñaron los métodos (e.g. Conocer el porcentaje de $hitrate$ obtenido). A este proceso se le llama cross-validation.

Veremos a continuación que este enfoque es la base de como usar los datos, pero que esta división no alcanza para obtener resultados totalmente confiables.

4.1.1. Reducción de over-fitting

Si decidieramos realizar nuestro entrenamiento de los métodos en base a una porción arbitraria de la base de datos etiquetada y usar el resto como evaluación, podemos correr el riesgo de caer en el **over-fitting**. Este fenómeno, no es otra cosa que entrenar nuestro método de manera demasiado sesgada y ajustada a un set de datos particular, donde predicciones a nuevos conjuntos de datos pueden no ajustarse o compartir la misma distribución.

Para paliar este fenómeno, no elegiremos arbitrariamente una partición de datos, sino que usaremos el método **K-fold**, que no es otra cosa que una *cross validation* pero dividiendo el conjunto de datos etiquetados en K particiones de igual tamaño y entrenar al método usando todas las combinaciones de elegir una partición como evaluación, y al resto como training.

Finalmente, se realiza un promedio de los resultados obtenidos en cada combinación elegida, para lograr tener un resultado fidedigno sobre el funcionamiento de nuestro algoritmo clasificador.

4.1.2. Herramientas utilizadas

Para generar estas K particiones desarrollamos un script llamado $makeInputTest.cpp$ en C++ que se encarga de tomar la base de datos etiquetada y dos clases de parámetros. Por un lado, los parámetros de entrenamiento, es decir, si se usará la base de datos reducidas o no, la cantidad de personas e imágenes por personas que se entrenarán, y la cantidad de componentes principales a tener en cuenta. Por otro, los parámetros de reconocimiento, es decir, cuántas personas serán reconocidas y cuáles. De ésta manera, para las personas que se reconocerán, se usarán las imágenes que no se usaron para ser entrenadas.

Para utilizar la herramienta, se debe ejecutar de la siguiente manera:
./tester **Reducidas/Normales(0/1) CantPersonas ImagenesPorPersona CantCompPpales CantPersonasAReconocer Cuales**

4.1.3. Generalidades

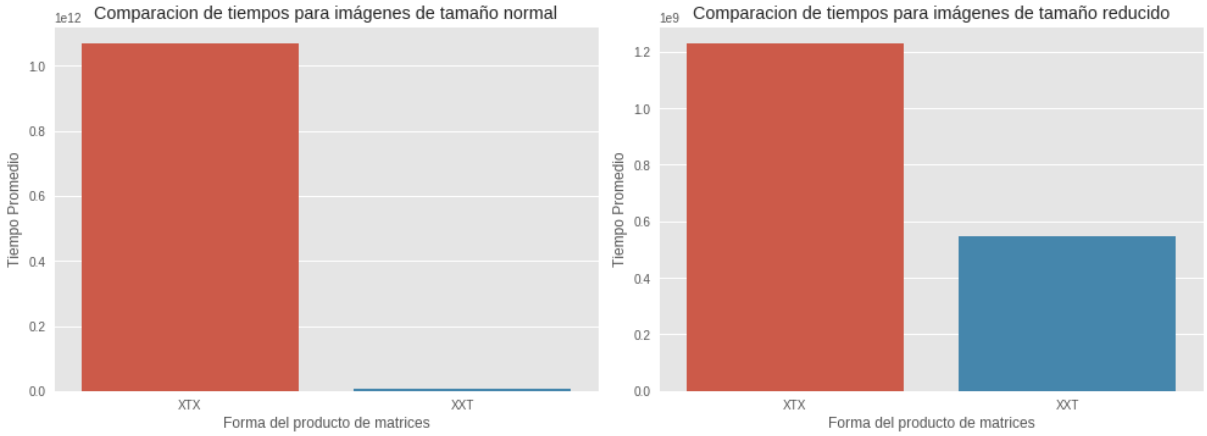
A la hora de experimentar, quisimos encontrar la configuración óptima para el reconocimiento de caras. Por un lado medimos los tiempos a la hora del entrenamiento, dado que existían como fue explicado anteriormente, dos variantes para armar la matriz de covarianza. Por un lado, generar M multiplicando $X * X^t$ y por otro $X^t * X$. Si bien en la precisión no influían, dado que la dimensión de la imagen resultante variaba considerablemente, influían en la cantidad de operaciones. Por otro lado, buscamos evaluar la calidad del reconocimiento de caras, según diferentes configuraciones. Para esto se usaron las métricas anteriormente mencionadas y buscamos variar, tanto la cantidad de componentes principales, como la cantidad de vecinos más cercanos a la hora de determinar sus K vecinos más cercanos. Además, vale aclarar que para el método de la potencia, se usaron dos criterios de parada distintos. El primero, el criterio que, si de una iteración a otra el valor del autovalor encontrado variaba menos que un **Epsilon**,

entonces no seguía buscando el autovalor. Y el segundo, si llegaba a la iteración 500. Luego será explicado cada experimento en particular, junto a sus conclusiones. Para calcular los tiempos, usamos la librería *Chronos* de **C++11**.

4.1.4. Experimento 1: Diferencia de tiempos de entrenamiento

Como fue mencionado en la sección anterior, para este experimento se buscó afirmar que existe una diferencia de tiempo a la hora de construir M con $X * X^t$ o $X^t * X$. Para esto, se hicieron dos pruebas, con las imágenes reducidas y con las imágenes normales. Para calcular el tiempo de entrenamiento, se entrenó con todas las imágenes (reducidas y normales) de la base de datos. El tiempo se empezó a calcular una vez que las bases de datos fueron leídas. Luego, se corrió el entrenamiento completo con 4 configuraciones distintas, la base de imágenes reducidas, con $M = X^t * X$ y con $\overline{M} = X * X^t$ y lo mismo con la base de imágenes normales.

Los resultados fueron los siguientes



Base de datos	XXT/XTX	Tiempo(ns)
Normal	XTX	1069096500000
Normal	XXT	7729010500

Cuadro 1: Comparación de tiempos para imágenes normales

Base de datos	XXT/XTX	Tiempo(ns)
Reducida	XTX	1229046000
Reducida	XXT	547551850

Cuadro 2: Comparación de tiempos para imágenes reducidas

4.1.5. Experimento 1: Conclusión

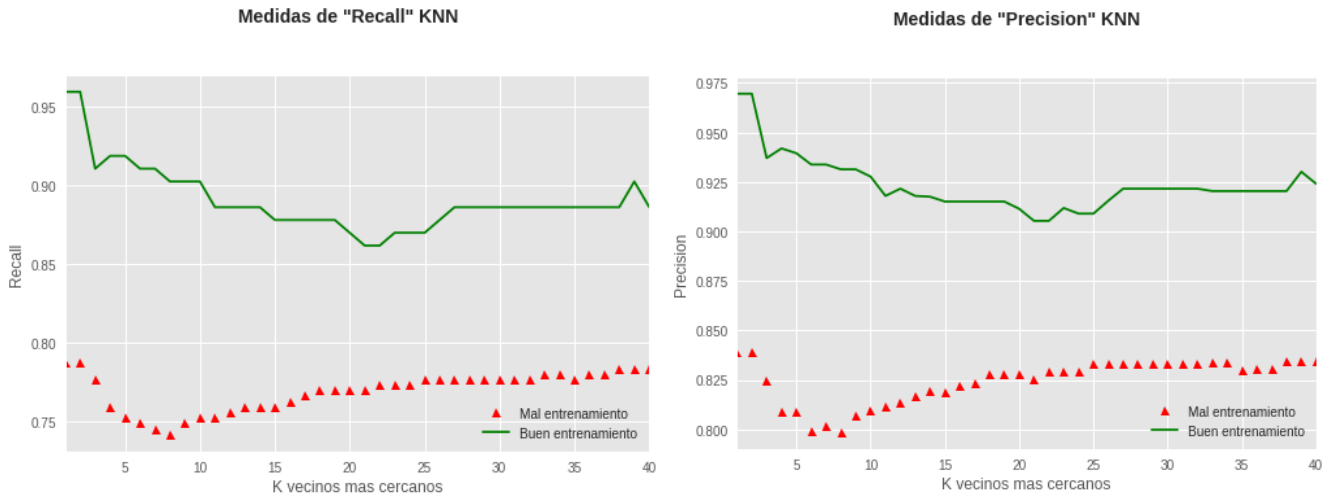
Luego, como pudimos observar, siempre que se usó $\overline{M} = X * X^t$ el tiempo fue notablemente inferior. Como explicamos anteriormente, dado que lo que optimiza esta manera de construir $\overline{M}$ son las dimensiones, y que esto va a ser óptimo, cuando la cantidad total de imágenes por persona al cuadrado sea menor a la cantidad de píxeles al cuadrado.

Luego, contamos con 10 imágenes por persona, y 41 personas. Esto nos da un total de 410 imágenes, luego $410^2 = 168100$ es la cantidad de elementos de $\overline{M} = X * X^t$. Por otro lado, en la base de datos reducida, la cantidad de píxeles es de $23 * 28 = 644$. Y en la cantidad de elementos sera $644^2 = 414736$, estos serán los elementos de $M = X^t * X$. Es decir, los resultados tienen sentido para la base de datos con la que contamos. Quedaría para futuras experimentaciones, realizar el mismo experimento para una base de datos donde hayan más personas y más imágenes, para de esta manera notar la ventaja de complejidad temporal de $M = X^t * X$.

4.1.6. Experimento 2: Calidad del reconocedor de caras para distintos K vecinos más cercanos

En el siguiente experimento, buscamos comparar para distintos **K** para **Knn**, cuánto influye estos en la calidad del reconocedor de caras. Como hipótesis, creemos que la comparación de una cara con todas las

imágenes de entrenamiento, dará un resultado más preciso, que obteniendo vecinos más cercanos. Para experimentar esto, se generaron dos clases de inputs con la base de datos de imágenes normales (ya que consideramos que la base de datos reducida, sería un subconjunto de esta y los resultados no variarían), la primera con un entrenamiento de 7 imágenes por persona y la segunda con 3. Además iteramos desde $K = 1$ hasta $K = 40$, incrementando en uno el K por cada iteración y así finalmente comparar la diferencia de calidad. Una vez obtenidos los resultados, se calcularon por cada K, sus verdaderos positivos (Vp), falsos negativos (Fn) y falsos positivos (Fp), métricas explicadas en secciones anteriores. Finalmente, se midió tanto la Precisión como el Recall. Los resultados obtenidos son:



4.1.7. Experimento 2: Conclusión

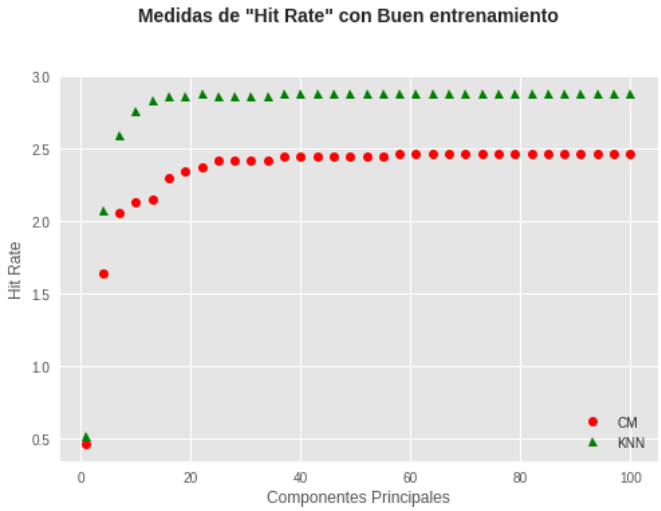
Como se puede observar en ambas métricas, para un buen entrenamiento, no influye de una manera determinante la cantidad de vecinos más cercanos a considerar. Tanto como para un buen o mal entrenamiento, resultó que la mejor calidad se consigue para $K = 1$. Esto afirma nuevamente, la primer conclusión donde el método de reconocimiento con más certeza es la del **Vecino más cercano**. Una razón que se cree causante, es que al utilizar $K = 1$, se estaría comparando cada imagen en la base de entrenamiento, dando resultados con mayor certeza acerca de qué imagen es la más cercana a la cara que se busca reconocer. Otra de las cosas que se puede observar de éste experimento, es el crecimiento de la calidad en ambas métricas para un mal entrenamiento. Si bien no mejora con respecto al mínimo K, existe cierta mejora a medida que crece el K. Se podría experimentar hacia donde tiende cuando K es mucho mayor a 40, y ver si existe mejora con respecto a $K = 1$. En la siguiente experimentación se hará una comparación de configuraciones como variar la cantidad de componentes principales, el método de elección y la calidad de entrenamiento.

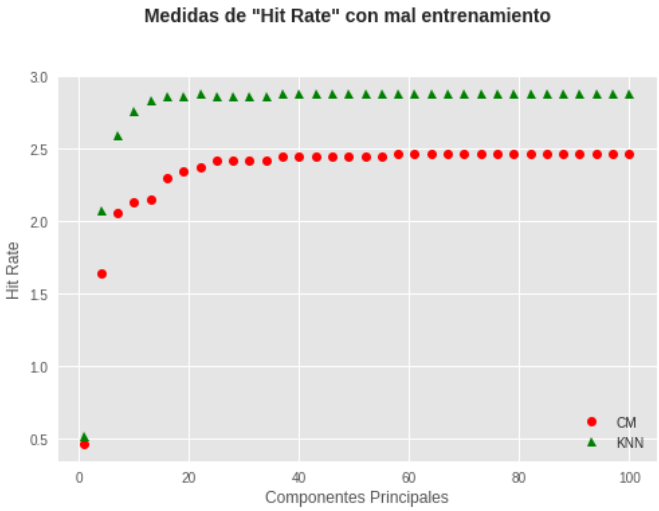
4.1.8. Experimento 3: Calidad del reconocedor de caras para diferentes Componentes Principales

En el siguiente experimento, quisimos observar, según el criterio de reconocimiento y los entrenamientos, cuál es la mejor configuración en cuanto a calidad. Como hipótesis suponemos, que es considerable la diferencia de calidad entre muchas componentes principales y pocas. Debido a que muchas componentes, generan un espacio más preciso y amplio para la comparación con las imágenes entrantes.

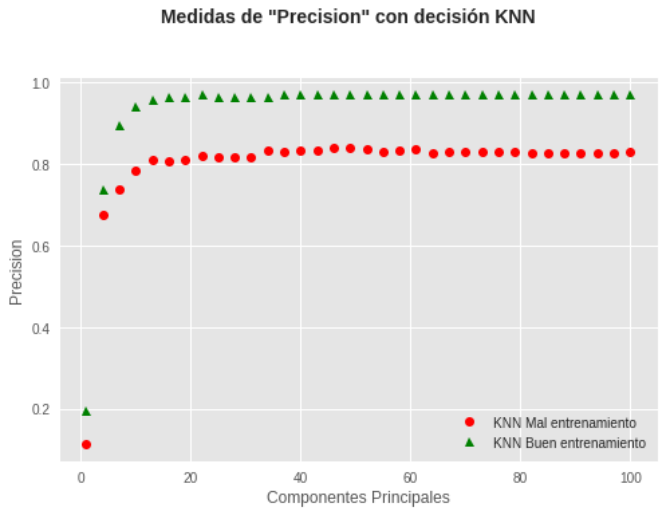
Además se fijaron dos criterios de reconocimiento, **Centro de Masa más cercano (CM)** y **K Vecinos más cercanos (KNN)**, con $k = 1$, dado que en el anterior experimento, arrojó como resultado parcial, que la mejor calidad se obtiene fijando $k = 1$. Al igual que el experimento anterior, se utilizaron dos entrenamientos, uno de 7 imágenes por persona (buen entrenamiento) y otro de 3 (mal entrenamiento). Luego, se iteró las componentes principales desde 1 hasta 100 cada 3. Esto se debió a que la diferencia de calidad entre *n* componentes principales y *n+1* sería despreciable. Luego por cada imagen a reconocer, se calculó por persona, sus **Vp, Fn y Fp**. Primero, se calculó la métrica de Hit Rate, con la particularidad que, como no tenía sentido comparar los diferentes entrenamientos, ya que la cantidad de casos para un entrenamiento u otro es diferente, y a mayor casos de prueba, menor dará el Hit Rate por como está

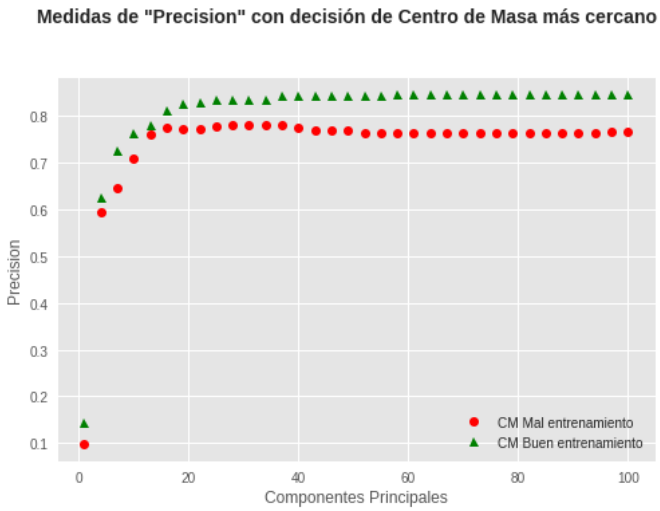
compuesta la métrica, por esto decidimos utilizarla para comparar únicamente los métodos de elección a razón de las componentes principales. Finalmente, se calculó tanto la Precisión y el Recall, comparando, cada uno de los entrenamientos. Luego los resultados del Hit Rate dieron,

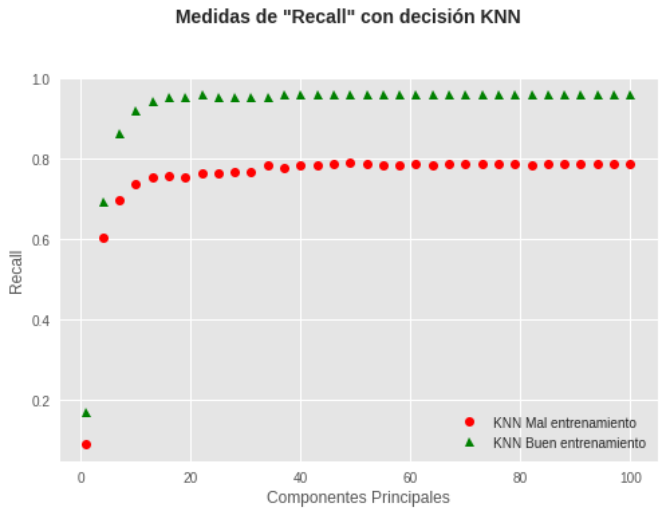


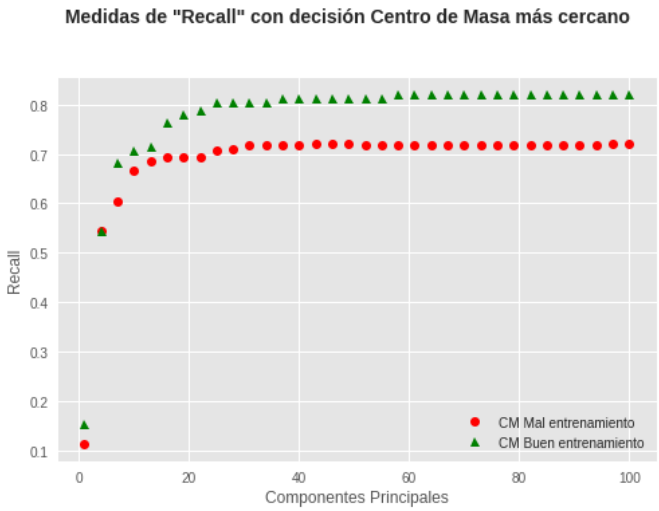


Mientras que para Precisión y Recall los resultados fueron









4.1.9. Experimento 3: Conclusiones

Como primer conclusión, observamos que el método de elección de **KNN**, con $K = 1$, es siempre superior al **CM**. Esto se podría deber a la precisión de cada uno. Ya que, por un lado **KNN**, compara imagen por imagen de la base de entrenamiento, hasta encontrar la más cercana a la nueva a ser reconocida. Pero por otro lado, **CM** toma un promedio de las distancias de las imágenes de cada persona. Teniendo el problema que, si una imagen de la persona *J* queda distante de todas las demás, aleja el **Centro de Masa** de donde realmente se concentran las coordenadas de la persona *J*. Si bien esta diferencia de precisión está, tambien se debe tener en cuenta, que la cantidad de comparaciones de **KNN** es igual a la cantidad de imágenes totales por cara a ser reconocida, mientras que **CM** hará un solo recorrido por todas las imágenes, para luego reducir la cantidad de información a una coordenada por persona, contando así con una ventaja temporal. Como segunda conclusión en cuanto a Hit Rate, notamos que la diferencia de calidad entre 0 y 10 componentes principales es considerable. Pero se reduce esa brecha aumentando cada vez más la cantidad de componentes.

También, para Recall y Precisión esta diferencia se mantiene, si bien existe un pequeño crecimiento de calidad, no es considerable. Luego no tenemos suficiente evidencia, como para afirmar que no existe un crecimiento considerable para mayores componentes principales. Pero podemos afirmar, que si se busca un resultado parcial, en poco tiempo, conviene fijar la cantidad de componentes principales entre 10 y 20, ya que para mayor componentes (menores a 50), no tendremos un salto calidad de reconocimiento. Y a mayor componentes, mayor tiempo de procesamiento (esto se puede ver, dado que a mayor componentes principales, más veces se deberá hacer el método de la potencia, además de que en la **tc** se deberán hacer también más operaciones).

Una segunda conclusión que encontramos es que lo que influye realmente en la calidad de reconocimiento, es el entrenamiento. Es decir, siempre que se tenga un buen entrenamiento, los resultados tenderán a ser óptimos.

5. Reconocimiento de una cara

Luego de trabajar con el entrenamiento y el reconocimiento de caras etiquetadas, se propone explorar la idea de si existe alguna manera de poder reconocer si una nueva imagen cualquiera puede ser reconocida como una cara o no. Con todas las restricciones que se puedan tener, pero con alguna aproximación más cercana, se propone identificar si la imagen que nos están pasando se la puede identificar como la foto de una cara humana con las mismas condiciones de aquellas que fueron utilizadas para realizar el entrenamiento (de frente, solo el rostro, etc).

Por lo tanto, imágenes de rostros deberían identificarse como positivos, y el resto de imágenes (animales, paisajes, etc), deberían ser negativos.

A lo largo del desarrollo del entrenamiento y el reconocimiento de imágenes de caras, se terminó desarrollando la idea de una ubicación espacial (en un espacio determinado y de ciertas dimensiones) para cada elemento, tanto del muestreo, como luego de los elementos a reconocer.

Se hizo hincapié en la idea de vecindad y cercanía de elementos pertenecientes a una misma clase de equivalencia. Esto último, fue producto de asumir qué imágenes de caras de una misma persona, comparten muchos rasgos (comparten el tono de piel, la distribución de las facciones, etc).

Extendiendo esta idea, pudimos verificar qué imágenes de caras de una misma persona (sacadas de una forma particular), suelen acumularse alrededor de un punto en el espacio (centro de masa de la persona).

Yendo más al fondo de la cuestión, dada la descomposición de las imágenes en sus componentes principales y el cálculo de los autovectores asociados, pudimos ver cómo es la representación de las imágenes en función de sus autovectores. Es decir, pudimos ver que son combinaciones lineales de sus α componentes principales, y que se genera un punto en el espacio, que es la interpretación de cómo una cara está explicada en función de las componentes principales de una base entrenada con imágenes de personas.

De esta manera, nos pudimos percatar que las características o atributos que comparten distintos rostros, son fácilmente identificadas con estos autovectores que representan la información principal de las caras. Y no lo es tan así, para poder explicar una imagen que corresponda a un paisaje o cualquier otro tipo de imágenes con diferencias importantes a los rostros.

Entonces volvimos a centrarnos en la idea de la vecindad espacial de los puntos, con un enfoque parecido al que utilizamos para identificar a qué persona corresponde una cara nueva.

La idea es ubicarnos en el punto medio de las coordenadas de todas las caras que utilizamos para entrenar a la base (el centro de masa de la base de entrenamiento) y generar una bola multidimensional para utilizar como clasificador de si una imagen es una cara o no.

Para calcular el centro de la bola utilizamos el promedio de todas las coordenadas calculadas. Asumiendo que no hay *outliers* en la muestra seleccionada. El problema de la existencia de outliers se puede solucionar fácilmente tomando un promedio α-podado (con las muestras ordenadas se sacan los $\alpha/2\,\%$ elementos más pequeños y los $\alpha/2\,\%$ elementos más grandes).

Luego calculamos la distancia de todas las imágenes de nuestra base de datos al centro de la bola para quedarnos con la máxima de todas, y así utilizar este parámetro como el radio de la bola.

Para realizar las pruebas, utilizamos la base de entrenamiento con 40 personas y 10 imágenes por persona. Dejando 10 imágenes para contrastar luego.

Sometimos a prueba nuestra hipótesis, y conseguimos verificar qué caras de sujetos que no se utilizaron para entrenar a la base, fueron reconocidas efectivamente como rostros de humanos. Además, imágenes de paisajes o animales fueron efectivamente reconocidas como imágenes que no pertenecían a rostros de personas.

Detallamos a continuación algunos *verdaderos negativos* (se obtuvo que no era un rostro y no lo es [Correcto]) y otros *falsos negativos* (se obtuvo que es un rostro y no lo es [Incorrecto]).

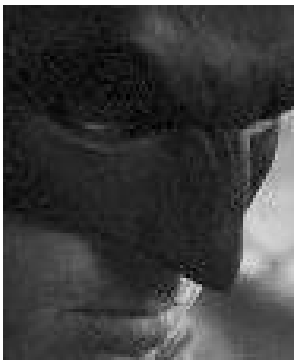


Figura 1: Correcto

Figura 2: Incorrecto

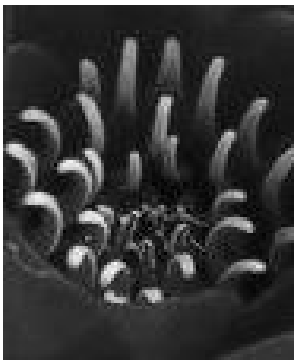


Figura 3: Correcto

Figura 4: Incorrecto

Figura 5: Incorrecto

Figura 6: Correcto

Figura 7: Correcto

Figura 8: Correcto

6. Conclusión General

Como conclusión general, notamos por un lado, la influencia del tamaño de la matriz M, para el tiempo de entrenamiento. Es necesario tener en cuenta las dimensiones con la que se operará, para tener tiempos considerables. En nuestro caso, al tener una matriz cuyo ancho y alto sean la cantidad de imágenes totales, en cuanto a tiempo, nos es mejor operar con la matriz $\overline{M}$.

Analizando los métodos de reconocimiento de caras, por un lado si se utiliza **KNN**, si bien requiere mayor tiempo, garantiza una excelente precisión si se utiliza con $K = 1$. El método es simple en cuanto a implementación y entendimiento y se puede analizar gráficamente sin ninguna dificultad.

Por otro lado, si se utiliza **Centro de Masa**, pensamos que una buena base de entrenamientos sin *outliers* tiene un centro de masa mejor distribuido entre las imágenes de las *caras* y una menor distancia a la más alejada del centro, lo cual genera una cota mejor para la decisión de si nuevas imágenes pertenecen o no a un rostro. El reconocimiento de rostros dio con un 100 % de hitrate, pero es factible que para una imagen de un rostro que no se encuentre en la base de datos, el programa resuelva que no es un rostro cuando éste se aleja bastante (mayor a la distancia máxima de las imágenes utilizadas como muestra) del centro de masa. Además, no pudimos analizar más profundamente por qué una cebra de perfil es interpretada como una cara, pero esto queda para futuros experimentos y un entendimiento superior de cómo son los autovectores principales de una base de datos de rostros y cómo puede representarse una cebra en función de los mismos.

Finalmente, si bien el entrenamiento influye en gran medida con la calidad, si no se cuenta con una base de datos con la cual entrenar, será necesario utilizar la mayor cantidad de componentes principales posibles, siempre y cuando dé el tiempo. De no ser así, con usar componentes principales entre 10 y 20, la calidad es considerable. De no tener el tiempo suficiente, el método de elección **CM**, si bien no garantiza una buena calidad dado la cantidad de operaciones, puede garantizar un buen tiempo. En nuestro caso, utilizamos para el reconocimiento de caras, 10 componentes principales.