DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Métodos Numéricos
Primer Cuatrimestre de 2018

Integrante	LU	Correo electrónico
Nicolás Ippolito	724/14	ns_ippolito@hotmail.com
Emiliano Galimberti	109/15	emigali@hotmail.com
Gregorio Freidin	433/15	gregorio.freidin@gmail.com
Pedro Mattenet Riva	428/15	peter_matt@hotmail.com

Contents

1 Resumen

Estudiaremos a lo largo de este informe diversos métodos que se utilizan para *clasificar* observaciones (es decir, dada una nueva observación y un conjunto de observaciones con diversas categorías, definir a qué categoría pertenece esta observación). En particular, evaluaremos la calidad (y el tiempo de ejecución) del algoritmo de clasificación `K Nearest Neighbors`, y posteriormente también de otro algoritmo llamado `K-puntos`.

Por otro lado, preprocesaremos nuestras observaciones utilizando el método `Principal Component Analysis`, el cual transformará nuestra base de datos en otra más pequeña y con otras características para luego poder aplicar nuestros algoritmos de clasificación.

Específicamente buscaremos clasificar caras de personas, o sea definir a qué persona le pertenece cierta cara. Luego, nuestras observaciones serán imágenes, todas en blanco y negro y con formato ppm.

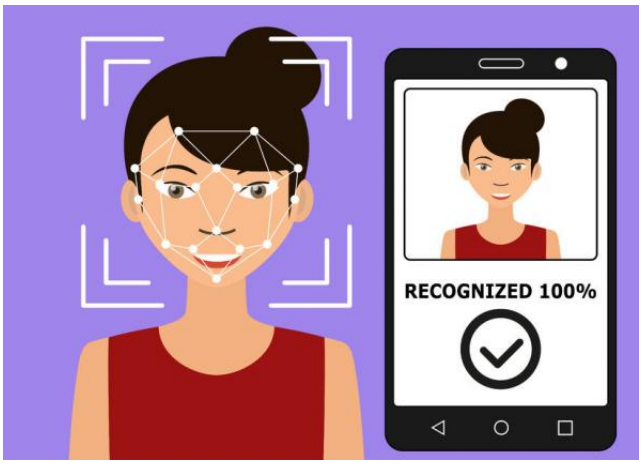
1.1 Palabras clave

Facial recognition, K Nearest Neighbors, Principal Component Analysis, Singular Value Decomposition.

2 Introducción

El siguiente documento explicará el problema a resolver con respecto al trabajo práctico número dos de la materia Métodos Numéricos, así como también explicará brevemente algunos conceptos teóricos necesarios para el entendimiento del informe, describirá los métodos utilizados y cómo estos fueron aplicados e implementados, y también expondrá diversos experimentos realizados junto a sus respectivas discusiones, conclusiones y resultados.

Lo que haremos será implementar un sistema de reconocimiento facial con varios métodos diferentes. El mismo será utilizado por las autoridades del emprendimiento *soja valley* -que revolucionará el país- con el fin de reconocer a todo trabajador/a que ingrese a las instalaciones pues allí hay mucha información confidencial (quién sabe qué puede generar esa información en las manos equivocadas... cuenta la leyenda que en el 'valle de la soja' hay secretos que podrían cambiar el rumbo del mundo entero).



Para esto, realizaremos un algoritmo que entrenaremos con una base de caras conocidas, para que luego pueda ser capaz de reconocer otra instancia de alguna de las caras de esta base (instancia que no haya sido utilizada para el entrenamiento), en otras palabras, definir de qué persona es esa cara.

3 Desarrollo

3.1 Conceptos teóricos

Los métodos numéricos que utilizaremos trabajarán con los **autovectores y autovalores** de una matriz. Luego, exponeremos la definición de éstos junto con los métodos para calcularlos, y posteriormente pasaremos a explicar los métodos para resolver el problema planteado en la introducción.

Definición

Un `autovalor` $\lambda \in \mathbb{R}$ de una matriz $A \in \mathbb{R}^{nxn}$ es un escalar que cumple $Ax = \lambda x$ donde x es un vector no nulo. Al vector x se lo denomina `autovector` asociado a λ. En términos gráficos, podríamos decir que los autovectores son vectores que sólo son estirados (o encogidos) cuando se les aplica la transformación lineal A.

Dada una matriz, ¿Cómo encontramos sus autovalores y autovectores?

3.1.1 Método de la potencia

Sea $A \in \mathbb{R}^{nxn}$ una matriz cuyos autovectores forman una base de $\mathbb{R}^n$, y además uno de sus autovalores (llamémoslo λ_1) tiene módulo estrictamente mayor que todo el resto, entonces puede demostrarse [2] lo siguiente:

Dado un vector $b_0 \in \mathbb{R}^n$ que necesite del autovector asociado a λ_1 para formarse, entonces la siguiente relación de recurrencia converge al autovector asociado a λ_1:

$$b_{k+1} = \frac{Ab_k}{||Ab_k||}$$

En la implementación, utilizaremos la cantidad de iteraciones como un valor de parámetro para que nuestro programa y el método termine. (Ver experimentación **4.2.1**)
A su vez, teniendo el autovector asociado v, λ_1 se puede calcular de la siguiente forma:

$$\lambda_1 = \frac{v^t A v}{v^t v}$$

Por lo tanto, obtuvimos el autovalor de mayor modulo y su autovector asociado.

3.1.2 Método de deflación

Ya sabemos cómo calcular el primer autovalor siempre y cuando la matriz A cumpla con ciertas condiciones. Ahora, veremos que si hay una relación de desigualdad estricta uno a uno entre todos los autovalores de A ($|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n|$) y además los autovectores forman un conjunto ortonormal (además de una base), podremos encontrarlos todos. Veamos las propiedades de la matriz $A - \lambda_1 v_1 v_1^t$, que formamos luego de aplicar el método de la potencia.

$(A - \lambda_1 v_1 v_1^t)v_1 = Av_1 - \lambda_1 v_1 (v_1^t v_1) = \lambda_1 v_1 - \lambda_1 v_1 = 0 v_1$
$(A - \lambda_1 v_1 v_1^t)v_i = Av_i - \lambda_1 v_1 (v_1^t v_i) = Av_i = \lambda_1 v_i$

Por lo tanto, la nueva matriz tiene autovalores $0, \lambda_2, \ldots, \lambda_n$ con autovectores asociados $v_1, \ldots, v_n$. Luego, podemos aplicarle el método de la potencia a esta nueva matriz, y seguir así sucesivamente hasta conocer los n autovalores y autovectores. Esto se llama **método de la deflación**.

Ahora, previo a esta introducción necesaria, pasaremos a explicar los métodos mediante los cuales implementaremos el sistema de reconocimiento facial.

3.1.3 Vecino más cercano

Buscamos, teniendo una base de diferentes caras con sus respectivas etiquetas (o sea, de qué persona es cada cara), ser capaces de determinar qué etiqueta le corresponde a una instancia nueva de alguna de las caras de la base.

Supongamos que todas las imágenes de caras son de tamaño 100x100 pixeles (un número arbitrario que servirá para facilitar la explicación). Se considera cada imagen como un vector $\mathtt{x} \in \mathbb{R}^n$, con $\mathtt{n} = 100 * 100$, por lo que cada una representaría un punto en el espacio euclídeo n-dimensional. Además, sabemos a qué clase pertenece cada una de

las imágenes de la base de datos (su *etiqueta*).

El método del **vecino más cercano** encuentra el punto `y` de la base de caras que más cerca se encuentra del punto que representa a la imagen que estoy buscando reconocer (`z`), y devuelve la etiqueta de `y`. Más precisamente, se debe recorrer toda la base de n vectores y buscar aquella que minimice la distancia euclídea:

$$\arg\min_{i=1,\dots,n} \; ||z - x_i||_2$$

3.1.4 K Vecinos más cercanos (KNN)

Bien podría pasar que, con el método previamente explicado, la etiqueta elegida no sea la correcta (esto lo veremos en la sección *Experimentación*).

Luego, el método puede generalizarse con el objetivo de aumentar su efectividad. Para ello, en lugar de tomar el punto que más cerca se encuentra del punto que representa a la imagen que buscamos reconocer, encontramos los `k` puntos más cercanos (con $\texttt{k} \in \mathbb{N}$). Finalmente, elegimos la moda de ese conjunto de `k` elementos (con respecto a su etiqueta), o en otras palabras la etiqueta que más se repite dentro del mismo. Exponemos su pseudocódigo:

Algorithm 1 K Nearest Neighbors

procedure K NEAREST NEIGHBORS(Base B, Imagen a)
Norma2Imagen ← CalcularNorma2(a)
VectorNormas ← Vector vacío

for $i \leftarrow 1$ to Tamaño(B) **do**
DistEuclideaI ← CalcularNorma2($a - B_i$)
$VectorNormas.Agregar(DistEuclideaI)$

Ordenar($VectorNormas$)
VectorNormasPrimerosK ← primeros `k` elementos de $VectorNormas$

return Moda($VectorNormasPrimerosK$)

El lector podría preguntarse "¿Qué valor de `k` conviene utilizar?", lo cual responderemos mediante diversos experimentos. Por ahora, notemos trivialmente que si tomamos $\texttt{k} = 1$ entonces el algoritmo no difiere en absoluto del explicado en **3.1.1**.

Sin embargo, hay un **problema** con esto: la *maldición de la dimensionalidad*. Al tener un gran número de dimensiones (recordemos que estábamos trabajando con dimensión 10000), por un lado el algoritmo podría volverse lento, y por otro la distancia euclídea entre el punto que representa a la cara que intentamos reconocer y todos los puntos de la base sería muy similar.

Diversas soluciones al obstáculo planteado fueron y son estudiadas por muchas personas. Una forma de encarar el problema es preprocesar las imágenes mediante la **reducción de dimensiones**.

3.1.5 Análisis de componentes principales (PCA)

El método **PCA** será utilizado para reducir la dimensión de los vectores que representan a cada imagen, quedándonos con aquellas componentes que más nos aportan en la distinción entre dos vectores diferentes, y además, asegurando que no haya ningún tipo de redundancia entre ninguna de las variables finales. Previamente, debemos dar algunas definiciones. Sean $x^1, x^2 \dots, x^n$ una secuencia de n datos, donde cada dato tiene m componentes (variables), definimos X como la matriz que tiene a x^1 como su primera fila, a x^2 como su segunda fila, ..., a x^n como su n-ésima fila (luego, $X \in \mathbb{R}^{nxm}$).

- **Media**: Se define la media como el promedio de un conjunto de números. En este caso, la media de todos los datos sería un vector $\in \mathbb{R}^m$ donde cada componente sería el promedio de los datos de cada columna de x. La notamos μ.

- **Varianza de una variable** x_j: Sea x_j una columna de X, la varianza mide cuán dispersados están los datos en general con respecto a la media. Mientras más varianza, más dispersos se encuentran los datos. La notamos $\sigma^2_{x_j}$
- **Covarianza entre dos variables** x_j **y** x_k: Sean x_j y x_k dos columnas de X, la covarianza es una medida que determina cuánto ambas varían de forma similar. Mientras más alta sea la covarianza, más relacionadas se encuentran (de la misma forma, que la covarianza sea cero indica que son variables independientes). La notamos $\sigma_{x_j x_k}$

Definimos a la matriz de covarianza M_x del conjunto de datos como:

$$M_x = \begin{bmatrix} \sigma^2_{x_1} & \sigma_{x_1x_2} & \cdots & \sigma_{x_1x_m} \\ \sigma_{x_1x_2} & \sigma^2_{x_2} & \cdots & \sigma_{x_2x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{x_1x_m} & \sigma_{x_2x_m} & \cdots & \sigma^2_{x_m} \end{bmatrix} \in \mathbb{R}^{mxm}$$

Adentrándonos un poco más, la matriz de covarianza se calcula con los siguientes pasos:

1. Calculamos la media de cada variable (μ_i es la media de la variable i). En otras palabras, es la media de cada columna de la matriz X.
2. Calculamos la matriz X', donde cada columna j de X' es $\frac{(X_j - \mu_j)}{\sqrt{n-1}}$
3. $M_x = (X')^t X'$

Notemos que si logramos, mediante una transformación de los datos, generar que la matriz de covarianza esté diagonalizada y que además la varianza sea lo más alta posible, entonces esto significará que no habrá redundancia entre ninguna variable (pues las covarianzas serán cero).

Sea V^t una matriz $\in \mathbb{R}^{mxm}$ que tiene los m autovectores de M_x en sus filas (que obtendremos mediante el *método de deflación*), se puede demostrar [1] que si transformamos los datos aplicando el cambio de base $(X')^t = V^t X^t$, la matriz de covarianza de los nuevos datos M'_x se encuentra diagonalizada, que es justamente lo que queríamos. Pero esto no es lo único, sino que también la diagonal de la nueva matriz de covarianzas estará ordenada de forma decreciente de arriba hacia abajo (es decir, el primer elemento de la diagonal es mayor o igual al segundo, y así sucesivamente). Esto implica directamente que la varianza de la primera componente (variable) es mayor o igual que la segunda, y así sucesivamente hasta recorrer toda la diagonal de la matriz. Por esto último, el método explicado también nos permitirá tomar sólo las α primeras componentes (siendo α un parámetro) pues estas son las que más aportan a distinguir los datos.

Algorithm 2 Principal Component Analysis

procedure PRINCIPAL COMPONENT ANALYSIS(Matriz de datos X) ▷ $X \in \mathbb{R}^{nxm}$
  $\mu_i \leftarrow$ Media de la columna X_i
  for $i \leftarrow 1$ to $n-1$ **do**
    $X_i \leftarrow \frac{(X_i - \mu)}{\sqrt{n-1}}$
  $M_x = X^t X$
  $V =$ MétodoDeflación(M_x) ▷ Asumo que el método devuelve en las columnas de V los autovectores de M_x
  $X \leftarrow VX$

  return X

Finalmente, habiéndole aplicado a los datos el método previamente explicado (**PCA**) y reducido la dimensión -quedándonos con lo que más nos importa de los datos-, podemos proceder a realizar las clasificaciones correspondientes mediante **kNN** (aplicándole también la transformación a la imagen que queramos reconocer).

3.1.6 Singular Value Decomposition (SVD)

Una descomposición matricial que puede ser aplicada cualquier tipo de matriz $\texttt{A} \in \mathbb{R}^{mxn}$ (en otras palabras, no necesariamente cuadrada), es la siguiente:

$$A = U\Sigma V^t$$

Donde, siendo r el rango (cantidad de vectores de la imagen de una matriz linealmente independientes) de la matriz `A`:

- $\texttt{U} \in \mathbb{R}^{mxm}$ es una matriz ortogonal cuyas primeras r columnas son los r autovectores de AA^t (normalizados) correspondientes a sus r autovalores de mayor módulo (notar que en realidad no hace falta tomar módulo pues AA^t es simétrica semi definida positiva, por lo que todos sus autovalores son ≥ 0).
- $\Sigma \in \mathbb{R}^{mxn}$ es una matriz diagonal. Sus primeros r elementos son los r autovalores más grandes de A^tA o AA^t (ordenados en forma decreciente) y los elementos restantes de su diagonal son todos cero. A los elementos mayores a cero se los denomina *valores singulares* de `A` y se los nota σ_i con $i = 1, \ldots, r$
- $V \in \mathbb{R}^{nxn}$ es una matriz ortogonal cuyas primeras r columnas son los r autovectores de A^tA (normalizados) correspondientes a sus r autovalores de mayor módulo (notar que en realidad no hace falta tomar módulo pues A^tA es simétrica semi definida positiva, por lo que todos sus autovalores son ≥ 0).

Además, las columnas de `U` y `V` serán cuidadosamente seleccionadas (pues los autovectores de una matriz no son únicos, como se puede ver en varios ejemplos en los cuales la multiplicidad de algún autovalor es mayor a 1), pues estarán relacionados por las siguientes fórmulas:

$$\begin{array}{ll} Av_i = \sigma_i u_i & i = 1, \ldots, r \\ Av_i = 0 & i = r+1, \ldots, n \\ A^t u_i = v_i \sigma_i & i = 1, \ldots, r \\ A^t u_i = 0 & i = r+1, \ldots, m \end{array}$$

Aclaración: Estamos asumiendo que los r autovectores normalizados (correspondientes a los r autovalores más grandes) de las matrices A^tA y AA^t forman un conjunto ortonormal (pues son columnas de las matrices ortogonales `V` y `U` respectivamente). Esto es correcto pues puede demostrarse que cualquier matriz simétrica tiene base ortonormal de autovectores.

3.2 Implementación

Los métodos numéricos explicados fueron implementados en lenguaje C++.

Por un lado tenemos una clase *Image* para almacenar cada imagen. En esta clase almacenamos el ancho y alto de cada imagen, sus elementos (con un puntero a uchar, similar a lo que hace el ppmloader al levantar las imágenes), y también guardamos la etiqueta de cada imagen mediante un string. Sin embargo, esta clase la utilizamos sólo para levantar cada imagen, pues para construir la base pasamos cada una de ellas a *vector<double>*, creando nuestra matriz final (aquella que representa a nuestra base, donde cada fila es una imagen) como un *vector<vector<double> >*.

Un detalle que necesitamos aclarar de la implementación concierne al método de la potencia. Al definirlo en secciones previas, dijimos que el método de la potencia se basa en la `convergencia` de una relación de recurrencia (o sea, con $k \to \infty$). Claramente, al estar trabajando en una computadora, debemos limitar k a algo finito. Luego, nuestro método tiene un parámetro de entrada *niter* que limita la cantidad de iteraciones que puede realizar el método (posteriormente experimentaremos sobre qué valores del mismo serán mejores).

Con respecto a *KNN*, cuando la moda de los k vecinos no es única (es decir, entre los k más cercanos a la imagen a clasificar no es una sola la categoría que más se repite), elegimos la etiqueta que primero haya completado el número máximo de votos obtenido. También expondremos más adelante las consecuencias de nuestra elección.

3.2.1 Algoritmos optimizados

En el método **PCA** trabajaremos con la matriz de covarianza X^tX (tendremos que encontrar sus autovalores y autovectores). Al tener una base de datos de aproximadamente 410 imágenes, donde cada imagen tiene $112 * 92$ pixeles en total, nuestra matriz `X` tendrá 410 filas y $112 * 92$ columnas (en otras palabras, $X \in \mathbb{R}^{410x112*92}$). Esto traerá como consecuencia directa que nuestra matriz de covarianza sea muy grande, pues tendrá dimensiones $112 * 92x112 * 92$, lo

cual ocasionará que el método de deflación tome un tiempo considerable en correr.

Sin embargo, la descomposición matricial explicada al final de nuestros conceptos teóricos (`SVD`) nos permiten acelerar muchísimo el proceso. Al saber cómo se relacionan los autovectores de X^tX con los de la matriz XX^t, podemos utilizar el método de deflación con XX^t -cuya dimensión será $410x410$-, obteniendo una mejoría temporal considerable. Una vez calculados los autovectores y autovalores de XX^t, obtendremos los que buscábamos originalmente aplicando una de las ecuaciones de *Singular Value Decomposition*: $\frac{X^t u_i}{\sigma_i} = v_i$.

Resta una importante aclaración: Tal cual fue explicado previamente, la ecuación es válida cuando $i = 1, \ldots, r$. Como al implementar tendremos que aprovechar la ventaja de *PCA* para quedarnos sólo con las α primeras componentes de cada imagen, entonces sólo calcularemos los α primeros autovectores y autovalores. Al tener la matriz de covarianza un tamaño muy grande, asumimos que su rango es mayor o igual que α (pues las probabilidades de que no haya al menos α columnas linealmente independientes en una matriz de estas características son casi nulas).

4 Experimentación y Discusión

En esta sección realizaremos una serie de experimentos para evaluar la calidad de los métodos utilizados, así como la variación (o no) del tiempo de ejecución según cómo vayan cambiando los distintos parámetros que utilizan los mismos. Para las mediciones de calidad de nuestros algoritmos de identificación de caras analizaremos las siguientes métricas: **Accuracy** (cantidad de clasificaciones correctas sobre el total de clasificaciones) y **F1-Score** (Media armónica que combina los valores de la precisión y recall).

La base de imágenes con la cual trabajaremos será la que fue propuesta por la cátedra (http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html), pero con un total de 410 imágenes (41 sujetos diferentes) tal cual fue entregada por la cátedra (la original tiene 400 imágenes de 40 sujetos distintos).

Por otro lado, utilizaremos el método **K-Fold cross validation** para dividir las imágenes entre *entrenamiento* y *testing*. Nuestra implementación de este se restringe a los valores de $K = [2, 5, 10]$, pues estos son los que permiten que todos los folds tengan exactamente el mismo tamaño. A su vez, nuestro código asegura que en cada uno de los folds las categorías estarán equitativamente distribuidas y esta distribución será aleatoria. Ejemplo: Si $K = 2$ (cada fold tendrá 205 imágenes), 5 imágenes de cada categoría irán a parar a entrenamiento y las 5 restantes a testing (donde esas primeras 5 imágenes serán elegidas al azar).

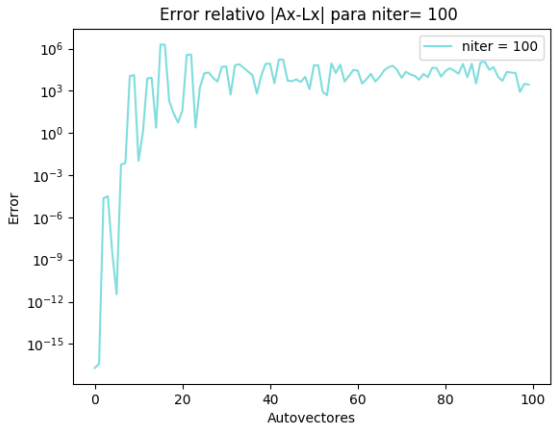
Todos los experimentos fueron realizados en una notebook con un procesador Intel Core I5-6200U y 8gb de memoria RAM.

Por último, de ahora en más notaremos k como el parámetro de `k nearest neighbors` y K como la cantidad de folds de `K-Fold`.

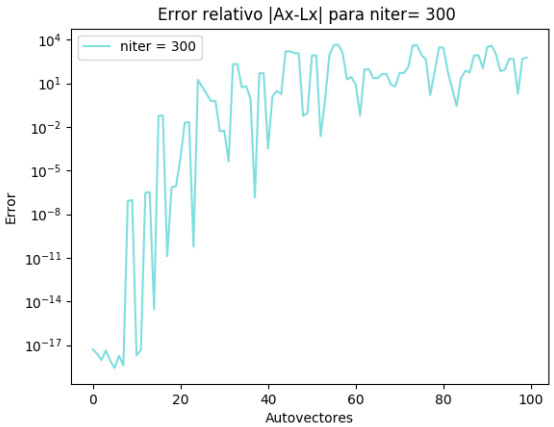
4.1 Número de iteraciones para el método de la deflación

Como vimos en la sección de *Conceptos teóricos*, necesitamos un parámetro para saber cuándo detener la iteración del **método de la potencia** y así concluir con un Autovector y un Autovalor con el menor error posible. Para esto definiremos para todos los experimentos un cierto número fijo iteraciones con el cual refinar los autovectores que buscamos conseguir.
Para determinarlo, realizamos la siguiente experimentación: Luego de varias pruebas con números mucho menores, se fijaron cuatro números finitos de iteraciones: 700, 500, 300 y 100. Llamaremos a este parámetro *niter*. Para cada uno de estos, y dada una base de imágenes aleatoria, se midieron los errores en escala logarítmica de $|AX_i - \lambda_i X_i|$, siendo λ_i y X_i los i-esimos autovalores y autovectores correspondientes para los primeros 100 vectores obtenidos por el método de la potencia.

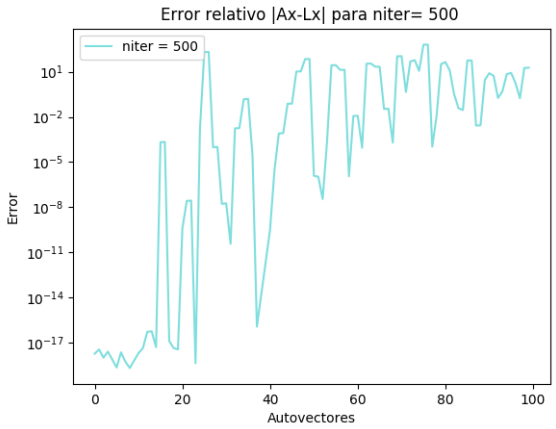
Resultados



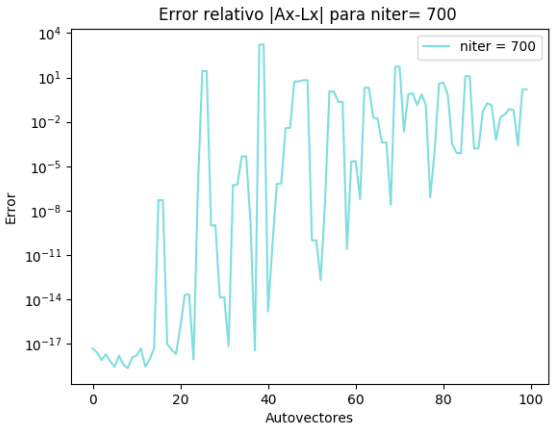
(a) Figura 1: 100 iteraciones



(b) Figura 1: 300 iteraciones



(a) Figura 2: 500 iteraciones



(b) Figura 2: 700 iteraciones

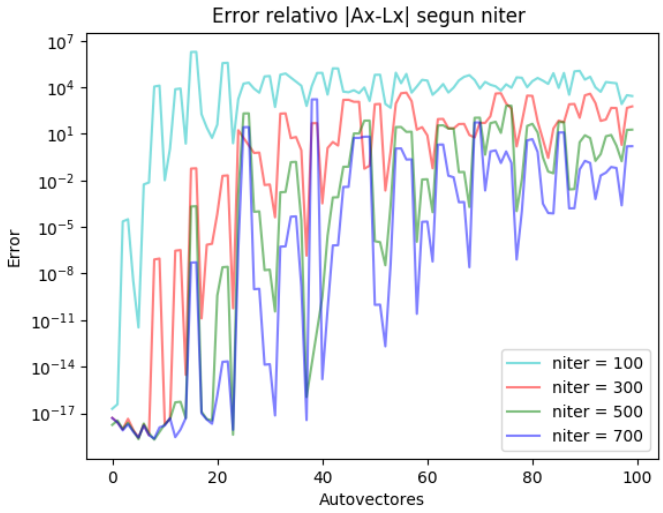


Figure 3: Comparación entre iteraciones

A partir de los resultados podemos observar:

Determinar los autovectores cuesta cada vez más iteraciones.
Se ve como a medida que se intenta conseguir más autovalores con el menor error posible, el número necesario de iteraciones para disminuir el error es cada vez mayor. Observar cómo para todos los gráficos, sin importar el *niter*, el error relativo tiende a crecer a medida que se buscan mayores autovectores.

A mayor cantidad de iteraciones, más perturbada es la función, aunque tiene menos error.
Los gráficos parecen más perturbados (en otras palabras, con mas picos, menos suaves) a medida que aumentamos las iteraciones. Este fenómeno se genera pues al aumentarse su posibilidad de tener outliers. Sin embargo, aun con estos, se puede ver en el gráfico de comparación final que a mayor cantidad de iteraciones se observa un menor error relativo, con lo cual obtenemos mejores autovalores y autovectores.

Los errores son considerables.
Variar entre 100 a 700 iteraciones posee un margen de diferencia considerable al querer minimizar el error relativo. Ya que con 100 iteraciones para casi todos los autovectores, su error relativo ronda los 10000, mientras que para 700 iteraciones, salvando outliers, su error relativo difícilmente supera los 10. Para números menores a 100 los errores llegaban a 10.000.000.000.

Finalmente veremos los tiempos de ejecución del método de potencia para obtener los 100 primeros autovectores, para cada niter evaluado anteriormente.

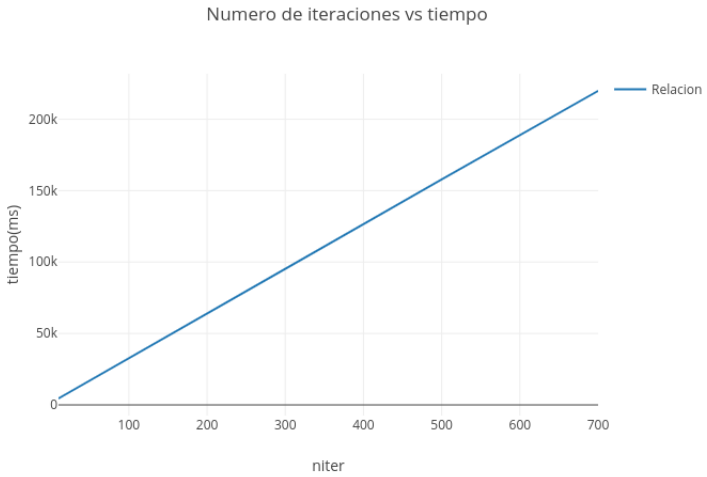


Figure 4: Comparación de tiempos

La relación lineal está clara, dado que la complejidad del algoritmo es constante, y depende únicamente de la cantidad de iteraciones pasadas como parámetro.

Dados estos resultados, decidimos utilizar ***niter*=700** para todos los experimentos donde se utilice el método de la potencia, por considerarlo óptimo en error relativo y pagando el costo temporal que implica.

4.2 Relación de k con el tamaño del conjunto de entrenamiento

En esta sección analizaremos cómo se relaciona k con el tamaño del conjunto de entrenamiento. Para la mayoría del siguiente análisis, nos abstraeremos de nuestra implementación de kNN con respecto al criterio de desempate cuando hay más de una etiqueta moda.

Nuestra base de datos en total posee 410 imágenes de 41 personas distintas. Todos los experimentos se hicieron con el método de K-fold. Para hacer que cada partición sea distribuida uniformemente (es decir, que haya la misma proporción de imágenes de persona por fold, lo que implica que también en la misma base de datos), tomamos $K = 2$, 5 y 10. Esto hace que el tamaño de cada fold sea 205, 82 y 41, y el tamaño del conjunto de entrenamiento 205, 328 y 369 respectivamente.

Sea C la cantidad de imágenes de cada persona por partición. Tomando K de esta forma, C puede tomar los valores de 5, 2 o 1, quedando así en la base de entrenamiento $10 - C$ cantidad de imágenes por cada persona.
Una vez fijamos la cantidad de folds (y por consecuente el tamaño de nuestra base de entrenamiento) procedemos a experimentar con el método Knn para resolver clasificadores.

(1) Supongamos que, para ambos métodos, los vectores Xi que más se asemejan a Z son aquellos que comparten su clasificación. Esto es, son imágenes de la misma persona.

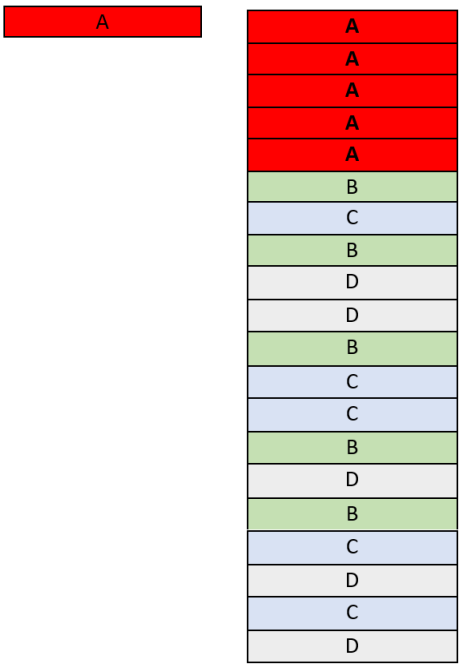


Figure 5: Imagen a etiquetar contra la base de entrenamiento ordenada por los primeros K vecinos.

De esta manera, los primeros C vecinos serán las imágenes con la misma etiqueta que Z, por lo que se necesitarán por lo menos $k = 2C$ vecinos para poder, eventualmente, empatar en la cantidad de votos (igualando la moda, haciendo que el calificador devuelva cualquiera de las dos personas)

Notar que las imágenes vecinas siguientes no necesariamente son de la misma persona. De ser así, entonces no podrían juntar los votos suficientes para modificar la moda, y el calificador seguirá devolviendo la persona correcta.

La única forma de asegurar que el calificador se va a equivocar, es si dentro de los k vecinos se encuentren más cantidad de imágenes de una persona distinta a la de Z.

Sin embargo, esto no es posible ya que tanto la base de entrenamiento como los folds fueron distribuidos uniformemente, impidiendo que haya más imágenes de una persona que de otra. Incluso si experimentamos con k = tamaño de la base de entrenamiento, el calificador resolverá eligiendo, de manera que esté implementada, cualquier etiqueta de persona.

Supongamos que, además de la condición anterior (1), nuestras particiones NO están uniformemente distribuidas. Si los folds no están distribuidos uniformemente, entonces nuestra base de entrenamiento tampoco lo está. Sea C_z la cantidad de imágenes de la persona Z que queremos identificar, C_i la cantidad de imágenes para cualquier persona i y C_j la cantidad de imágenes para una persona j, de tal manera que $C_{i_1} \geq C_{i_2} \geq \cdots \geq C_j$ y $C_j > C_z$ (en otras palabras, C_j es el menor de los C_i tal que sea mayor que C_z), todas en la base de entrenamiento.

Con seguridad podemos afirmar que una vez tomado $k = C_j * 41$ nuestro calificador va a fallar, dado que entre esos k se consiguieron al menos C_j imágenes de alguna persona, modificando la moda y devolviendo una etiqueta diferente a la que queríamos (Z). De todas formas, es posible que exista un $k < C_j * 41$ donde ya la moda haya sido modificada.

A su vez, eventualmente dependiendo de la distribución, es posible que $C_z > C_i$ para toda i persona. En ese caso podemos asegurar que nuestro calificador *nunca* va a fallar. Esto se debe a que ninguna otra persona podrá superar ni igualar los votos suficientes como para modificar la moda dominante de la persona Z, devolviendo el resultado correcto sin importar el k seleccionado.

En ambos casos, se puede ver que a medida que aumenta el k también se incrementan las probabilidades de que el calificador falle, sea porque la media cambia totalmente o tiene más posibilidades de equivocarse si todos tienen los mismos votos (aunque esto no es cierto en nuestra implementación por lo que veremos más adelante)

Ahora supongamos que, para ambos métodos, los vectores Xi que más se asemejan a Z *no necesariamente* son aquellos que comparten su clasificación. De esta manera puede pasar que los primeros C vecinos no sean de la misma persona que el vector Z. Por consecuente no podemos describir exactamente cómo es el comportamiento del clasificador, ya que no podemos predeterminar cuántos vecinos se necesitan tomar para contemplar el total de imágenes con la misma persona.
Si bien se presume que los vecinos mas cercanos van a coincidir con la persona Z, puede que existan pequeñas excepciones que modifiquen nuestro cálculo y le agreguen un margen de error.

Finalmente respecto a nuestra implementación
En caso de *empate de modas*, nuestro calificador devolverá la persona que **primero logre alcanzar el máximo número de votos** y por consecuente la moda.
A su vez, como nuestras particiones para experimentar/testear están distribuidas uniformemente, nuestro calificador debería tener muy alto nivel de accuracy, pues si para cierto k todas las imágenes de la etiqueta correcta (que hay en nuestro conjunto de entrenamiento) estaban entre los k más cercanos, aumentar el valor de k no debiera causar diferencia alguna en el resultado del clasificador. Sin embargo, si bien presumimos que los vecinos más cercanos van a coincidir con la persona de la imagen que buscamos calificar, puede que la performance vea un sesgo de error al momento de hacer las experimentaciones, ya que pueden existir otras imágenes con distintas etiquetas de personas más parecidas a la imagen a calificar que otras imágenes de su misma persona.

Para justificar la validez (o no validez) de esta última hipótesis, realizaremos diversos experimentos en las próximas secciones.

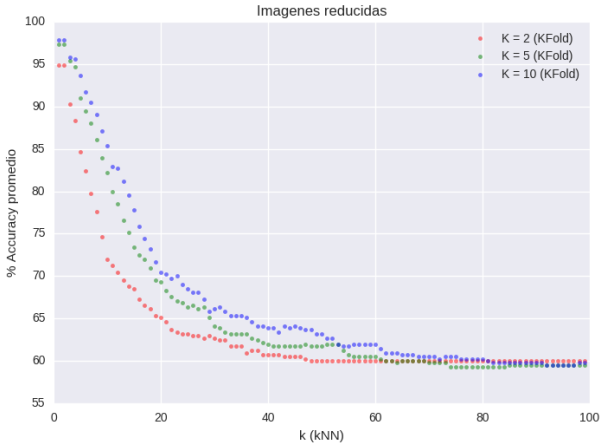
4.3 Clasificando con KNN

Comenzaremos midiendo la `accuracy` de nuestro clasificador utilizando únicamente el método **K-Nearest Neighbors**, y cómo ésta varía a medida que el k (fijado entre 1 y 100) va aumentando. Fueron empleadas las imágenes de la base reducidas (las entregadas por la cátedra en la carpeta 'reduced') con los tres valores K especificados previamente. Así mismo, también fue realizado el experimento con las imágenes de la base sin reducir, pero sólo con $K = 5$ pues clasificar una de estas con este algoritmo (sin preprocesar) tiene un tiempo de ejecución alto, por lo menos con nuestra implementación del mismo.

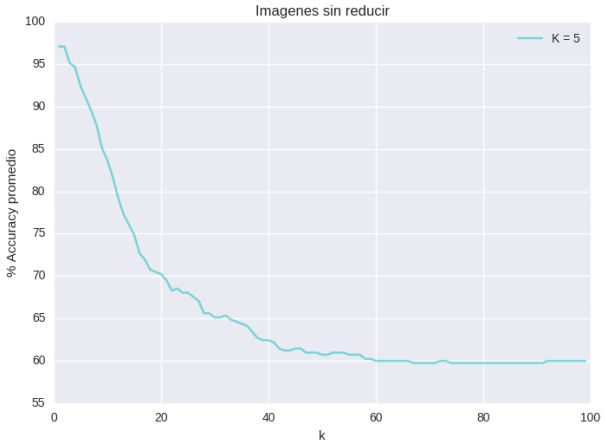
La medición de `accuracy` final es la suma de las accuracys de cada iteración de K-Fold dividida por K.

Como ya hemos justificado al final de la sección **4.2**, creemos que dada nuestra base de datos perfectamente distribuida (con respecto a cantidad de instancias de cada etiqueta) y nuestra implementación del algoritmo, tendremos una precisión relativamente alta, que irá disminuyendo a medida que aumenta k pero a partir de cierto valor de este, no variará con su crecimiento.

Resultados



(a) Figura 6: Imágenes reducidas



(b) Figura 6: Imágenes originales

Observamos que en ambos casos nuestra hipótesis pareciera cumplirse: Mientras más crece el k, menor es la

`accuracy`. Sin embargo, cuando k llega aproximadamente a aproximadamente 60, las clasificaciones mantienen la misma precisión (ya hemos explicado previamente por qué ocurre este fenómeno). A su vez, puede verse que cuando k es muy bajo la accuracy es muy alta, lo que indicaría que probablemente en nuestra base de datos las primeras imágenes más cercanas son las de la etiqueta correcta. En otras secciones venideras seguiremos experimentando para ver si efectivamente lo dicho en el último párrafo, que ya había sido presumido en la sección **4.2**, es válido.

Analizando la **Figura 1-a)** también podemos estudiar la calidad del método según K. Se puede observar que cuando $K = 10$ la accuracy de KNN es mayor, aunque no hay una diferencia muy grande con respecto a $K = 5$. Esto tiene sentido pues para estos dos parámetros en cada iteración de KFold hay una o dos imágenes (respectivamente) correspondientes a cada etiqueta en el conjunto de testeo, por lo que las 9 u 8 restantes se encuentran en el conjunto de entrenamiento. Luego, nuestro clasificador tendrá más datos para entrenarse, aumentando las chances de devolver el resultado correcto.
Todo lo contrario cuando $K = 2$, ya que habría 5 imágenes de cada etiqueta en cada uno de los conjuntos, disminuyendo la cantidad de datos que tendrá nuestro clasificador para entrenarse, y por ende reduciendo las chances de que la imagen sea clasificada correctamente, lo cual es correlativo con los resultados del experimento donde se ve claramente que la accuracy es más baja cuando $K = 2$.

Aún queda una cosa más por estudiar. Tal cual fue explicado en los conceptos teóricos, KNN es afectado por la *maldición de la dimensionalidad*. Compararemos la accuracy de las imágenes reducidas con las originales ($K = 5$) para ver si nuestra base de datos es afectada por este problema. Creemos que debiéramos ver una mejor accuracy en las imágenes reducidas que en las originales.

Resultado

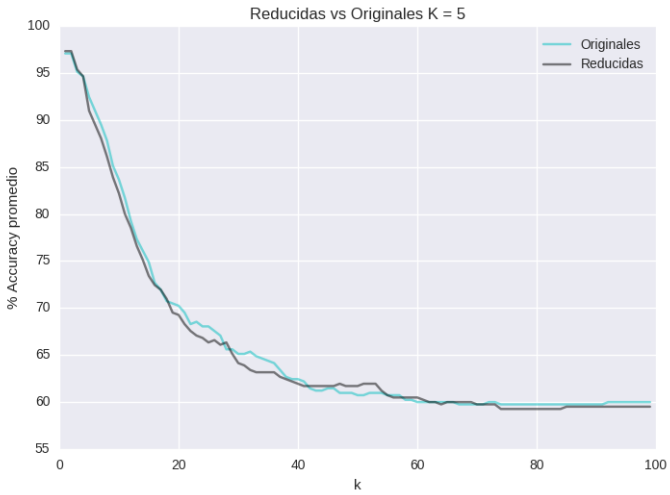
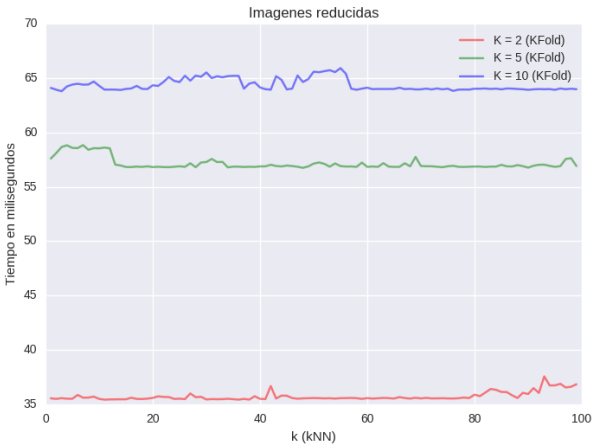


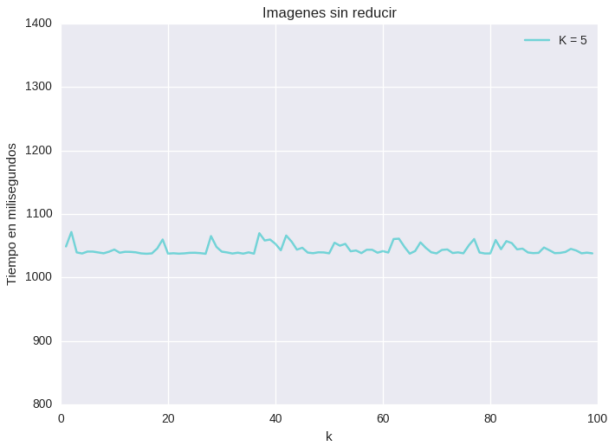
Figure 7: Imágenes reducidas vs originales con $K = 5$

Esta vez nuestra hipótesis estuvo errada -por lo menos con $K = 5$-, ya que se puede observar en la **Figura 7** que hay muy poca diferencia en la accuracy, y que incluso para algunos k las reducidas funcionan un poco mejor que las originales y para otros es al revés. Esto puede deberse a que el desempeño del método con las imágenes reducidas se vea afectado por la pérdida de datos generado al reducir cada imagen.

En cuanto al **tiempo de ejecución**, medimos lo que tarda en promedio clasificar una imagen para cada valor de k (o sea, la suma del tiempo que tardaron todas las clasificaciones que hubo en cada iteración de K-fold, divididas por: $size(conjuntoTesting) * K$).



(a) Figura 8: Imágenes reducidas



(b) Figura 8: Imágenes originales

Se puede observar en las **Figuras 8-a) y b)** que el parámetro k no afecta en absoluto el tiempo de ejecución, ya que al graficar cómo cambia el tiempo según la variación de este parámetro, la función es prácticamente constante. A su vez, queda claro al comparar los milisegundos entre las imágenes reducidas y las no reducidas ($K = 5$) que, tal cual habíamos expuesto al principio de la experimentación, el tiempo de ejecución es efectivamente bastante más grande al clasificar una imagen no reducida.

A su vez, se ve que el valor de K sí modifica el tiempo de ejecución, lo cual lógicamente debe ocurrir ya que este valor cambia los tamaños del conjunto de entrenamiento y de testeo.

4.4 Clasificando con KNN + PCA

En esta sección mediremos la `accuracy` del clasificador de imágenes aplicando el método **PCA**. En secciones anteriores explicamos cómo esta técnica planea resolver la *maldición de dimensionalidad* con la que se enfrenta KNN, permitiéndonos además quedarnos con los pixeles más importantes (es decir, que más información aportan) al momento de resolver una etiqueta.

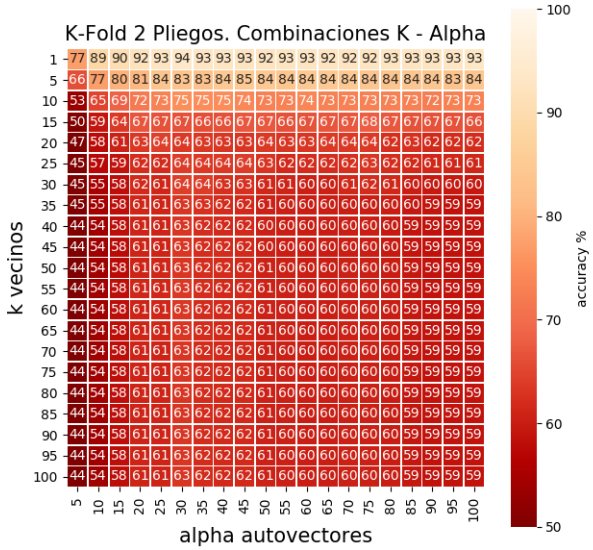
Con nuestra experimentación intentaremos identificar la mejor combinación entre los parámetros k y *alpha* (variando ambos entre 1 y 100) para aplicar los algoritmos KNN y PCA respectivamente, prestando atención también a los tiempos de ejecución de cada clasificación.

Verificaremos también si variando el K (en otras palabras, cambiar las proporciones entre las bases de entrenamiento y de testing) se produce algún cambio.

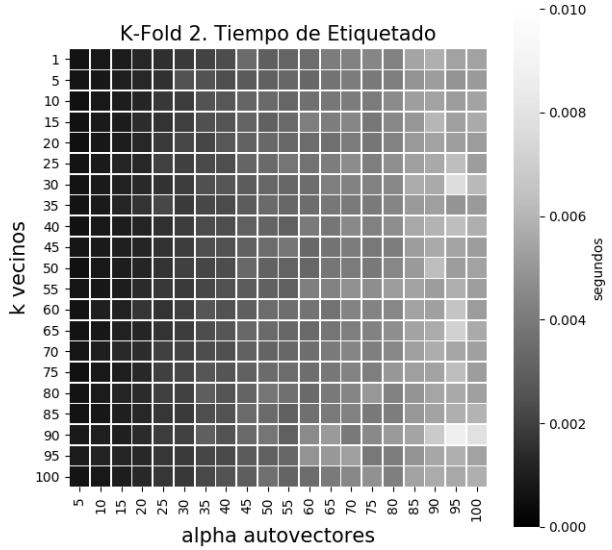
Como primera hipótesis creemos que se va a mostrar un mejor resultado para los casos donde *alpha* se encuentra 'en el medio', pues si utilizamos un *alpha* muy grande volveríamos a caer en el problema de la dimensionalidad, y para *alphas* muy pequeños estaríamos tal vez considerando muy pocos píxeles, menos de los que podrían ser considerados una cantidad relevante.

Luego, con respecto al factor cantidad de vecinos, creemos que (tal cual fue expuesto en secciones anteriores) cuando este se empieza a pasar de la cantidad de imágenes de cada label que tenemos en la base de entrenamiento, la accuracy disminuirá ya que se empezarán a clasificar imágenes de más, lo que aumentará las probabilidades de seleccionar varias imágenes de otro label. Sin embargo, a partir de cierto k no debiéramos ver mucha reducción con respecto a esta métrica.

Los primeros **resultados** obtenidos fueron:



(a) Figura 9: Accuracy



(b) Figura 9: Tiempo de ejecución

Como podemos ver en la **Figura 9-a)** los resultados en cuanto a la variable *alpha* fueron un poco menos significativos que los esperados, pues observamos que a medida que aumentamos el *alpha* la accuracy no se ve afectada prácticamente en nada.

Sin embargo, vemos por la **Figura 9-b)** que en los tiempos de ejecución sí hay una diferencia marcada: si prestamos atención, para los *alphas* entre 30 y 50 la accuracy se mantiene en valores casi idénticos que para $alpha = 100$, pero en cuanto a los tiempos hay una diferencia significativa sobre cuánto nos tarda clasificar cada imágen.

Luego, aunque el movimiento de la variable *alpha* no marque grandes resultados en cuanto a la precisión de la clasificación, sí marca una gran diferencia en el tiempo que toma la misma, por lo que concluimos que el **alpha ideal** debería ser $alpha = 40$, ya que para *alphas* más pequeños se puede notar una leve diferencia en accuracy y prácticamente nula en cuanto a tiempos, y con respecto a *alphas* mayores, se ve principalmente una gran diferencia de tiempos y no en accuracy.

Por el otro lado tenemos para analizar la variable k vecinos, sobre la cual podemos notar observando el gráfico de tiempos que esta no tiene un gran efecto en cuanto al tiempo de clasificación. Ya dejamos en claro que esta métrica se relaciona casi directamente con el *alpha* de PCA tomado, y no tiene relación con k (como también veíamos previamente en experimentos previos).

Sin embargo podemos ver que afecta sustancialmente a la precisión de las predicciones. La mayor diferencia la notamos hasta $k = 20$, y esto se debe a nuestra implementación del algoritmo KNN, como hemos explicado previamente. Luego, esta experimentación también verifica nuestra hipótesis planteada al final de la sección **4.2**, que también parecía validarse a lo largo de los tests de KNN sin PCA.

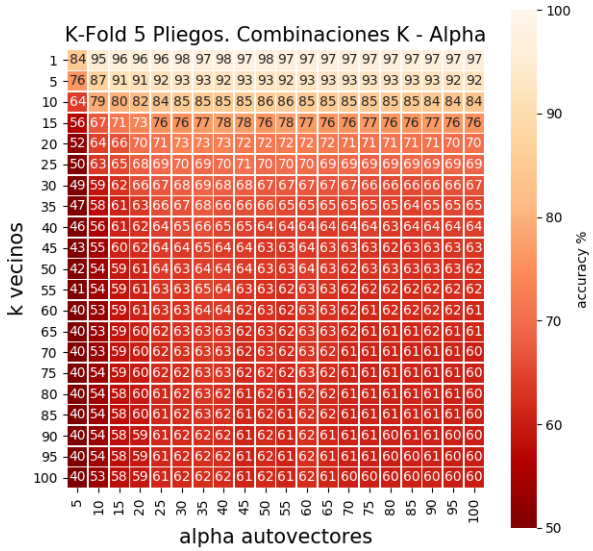
Ahora, al igual que para la variable *alpha*, querremos tomar un **k óptimo**, que según los resultados, vemos que la premisa actual es que la primer imagen suele ser la correcta, mostrando en general un 93% de aciertos. Luego, tomaremos por ahora a nuestro k óptimo como 1.

Para los siguientes gráficos que vamos a mostrar, queremos realizar la misma experimentación pero para un K mayor, variando así las proporciones entre las bases de testeo y de entrenamiento. Decidimos realizar los mismos tests pero fijando $K = 5$ (nos parece un poco redundante hacer lo mismo también con $K = 10$ pues la cantidad de imágenes de cada label en entrenamiento y testing es bastante similar con estos dos valores de K).

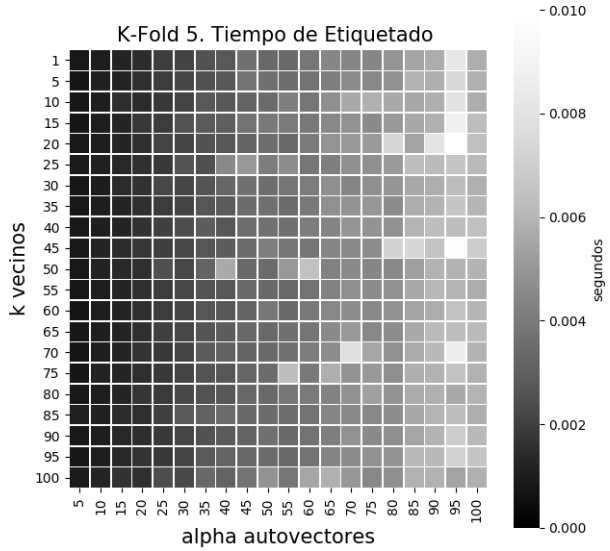
Como hipótesis para la siguiente experimentación esperamos en cuanto a accuracy alguna leve mejora, ya que en este caso a diferencia del anterior, vamos a trabajar con una base de entrenamiento mayor a la base de testeo.

Sin embargo, esperamos que la forma en que cada variable se relaciona con cada métrica no cambie, pues pensamos por el análisis anterior (y el de secciones previas) que este comportamiento es más general y por lo tanto no es provocado por los tamaños de las bases.

Los **resultados** obtenidos son:



(a) Figura 10: Accuracy



(b) Figura 10: Tiempo de ejecución

Como podemos ver, obtuvimos resultados muy similares a los esperados, efectivamente notamos los mismos comportamientos de las variables k y $alpha$, sobre las métricas accuracy y tiempo.

En particular podemos salvar que todavía $alpha = 40$ Y $k = 1$ siguen siendo los valores óptimos para nuestro clasificador de imágenes. A su vez,vemos que no se demuestra ninguna relación entre la precisión del sistema con respecto a los tamaños de las bases.

4.5 Métricas Adicionales

4.5.1 F1Score

Además de las métricas ya evaluadas, como la tasa de eficacia (accuracy) y el tiempo de ejecución, hay alternativas para intentar definir el nivel de éxito que tiene el algoritmo con el cual se entrena el predictor de imágenes. Entre ellas, podemos analizar el puntaje F_β. Este se calcula con la siguiente fórmula:

$$F_\beta = (1 + \beta^2) \times \frac{precision \times recall}{\beta^2 \times precision + recall}$$

En nuestro caso, analizaremos el puntaje F_1. Las variaciones en el valor de β alteran cuál de las dos variables ($precision$ o $recall$) se evalúa con mayor relevancia. Un β mayor a uno aumenta la proporción con la cual $precision$ va a modificar el denominador ($precision + recall$), dándole mayor peso a dicha variable. Viceversa, para β entre 0 y 1. Pero como para la siguiente experimentación buscamos analizar $precision$ y $recall$ con igual importancia, se tomo β como 1.

Mencionamos que una diferencia muy importante entre esta métrica de comparación y $accuracy$, es que F_1 funciona de manera binaria. Donde antes un caso positivo era simplemente una muestra correctamente etiquetada, al analizar de manera binaria, se debe seleccionar una muestra en particular. Con esta se parte en cuatro casos distintos los resultados de correr el predictor sobre un vector de archivos de testeo. Dada una etiqueta de imagen X, estos cuatro casos son los siguientes:

- tp: cuando el predictor analiza una imagen de la clase X y este devuelve la etiqueta X.
- fp: cuando el predictor analiza una imagen que no es de la clase X y este devuelve la etiqueta X.
- fn: cuando el predictor analiza una imagen de la clase X pero este devuelve una etiqueta distinta a X.

- *tn*: cuando el predictor analiza una imagen que no es de la clase X y este devuelve una etiqueta distinta a X

Esto nos plantea una dificultad: Al tener nuestras muestras para entrenar al predictor distribuidas de tal manera que por cada etiqueta distinta hay solo 10 muestras, para que el conjunto de test sea balanceado deberá tener como máximo 20 items. Si quisiéramos crear un conjunto de testeo de solo 20 elementos, entonces a lo sumo 10 serían nuestros casos positivos (dada una etiqueta X elegida) y los otros 10 pueden ser cualquiera de las otras 400 muestras. Pero esto ni siquiera tiene en cuenta el conjunto de testing, el cual claramente necesita muestras de la etiqueta X para obtener resultados positivos. Sin embargo, de repetir muestras tanto en el entrenamiento como en el testeo, estaríamos en un caso de $overfit$. Una alternativa sería separar las 10 muestras en mitades, pero seguiríamos estando escasos de muestras tanto para entrenar como para testear.

Aún así, decidimos evaluar el puntaje F_1 con el propósito de analizar como éste va cambiando a medida que se aumenta la cantidad de vecinos tomados en cuenta en el algoritmo de knn (k) y la cantidad de autovectores con los cuales se define la matriz cambio de base de PCA ($alpha$). Se iteró sobre las 41 etiquetas distintas del set de imágenes otorgado por la cátedra, analizando el puntaje F1 tomando cada una como una etiqueta positiva y el resto como lo contrario.

Se espera que, dado que los conjuntos de test sobre los cuales evaluamos el puntaje F1 no están bien distribuidos para cada etiqueta X cuando se la analiza binariamente, entonces es probable que los resultados arrojen resultados aún mayores que la *accuracy*. También se espera que al igual que en los tests anteriores, el resultado máximo se halle al tomar $k = 1$ y un $\alpha = 40$. Los resultados obtenidos fueron:

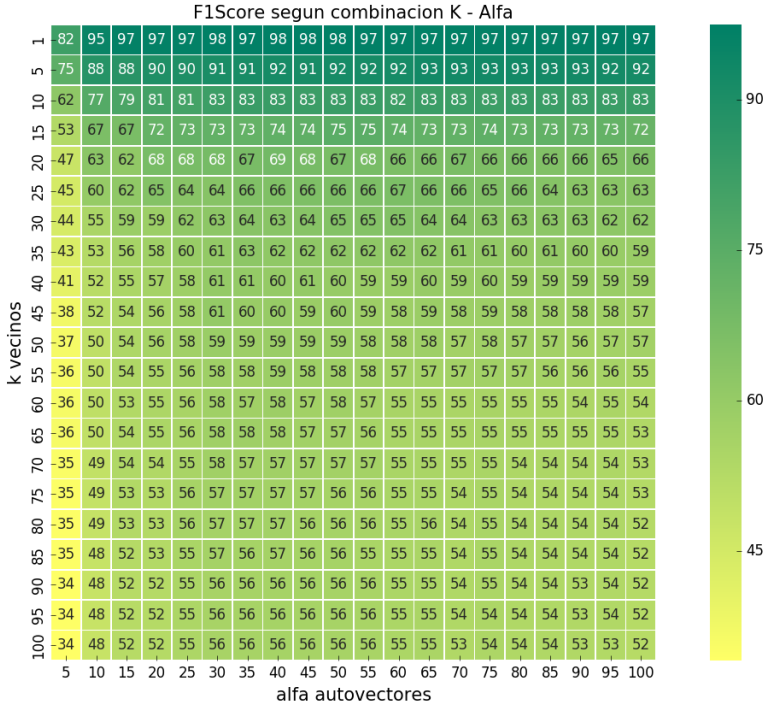


Figure 11: F1Score

Como se puede ver, de acuerdo a las expectativas indicadas previamente, obtuvimos resultados muy similares a los esperados. Efectivamente notamos los mismos comportamientos de las variables k y α sobre las métricas accuracy y tiempo. Tanto como se analizó previamente, para todos los valores del eje x, hay un valor de k vecinos a partir del cual el puntaje F_1 deja de reducirse. Una diferencia que podemos hallar, es que el puntaje F_1 es mas severo, viendo como para $\alpha = 1$ y $k = 100$, el mínmo alcanzado fue 34, a diferencia de 44 como con el acurracy.

En conclusiones generales, se puede afirmar que siguiendo los resultados ya presentados, todavía $alpha = 40$ Y $k = 1$ siguen siendo los mejores valores óptimos para nuestro clasificador de imágenes. De todas formas, como se mencionó antes, en nuestra opinión esta prueba no aporta una métrica confiable, ya que para cada iteración del KFold el conjunto de testeo solo contiene una imagen de la clase siendo evaluada como "caso positivo". Sabiendo que cada conjunto de test tiene 41 imágenes por cada KFold, esto nos dice que cada vez que se calculo el puntaje F_1, fue hecho con una base de imágenes mal distribuida (solo 2,5 % de casos positivos).

4.5.2 Matriz de Confusión

La matriz de confusión es otro modelo con el cual se puede describir la performance de un algoritmo de clasificación, en nuestro caso, de imágenes. Como sucede en nuestra experimentación, es necesario que las etiquetas de las imágenes pasadas por el predictor sean conocidas, ya que la matriz revela para cada elemento sabiendo su etiqueta, si se le asignó entonces una correcta, o en el caso contrario, con cual se la confundió. En una situación de entrenamiento no supervisado, este análisis sería imposible de realizar.

De esta manera se define la siguiente matriz de $n \times n$, donde n es la cantidad de clases de imágenes distintas. Para uno de los índices, esta representará el número de etiqueta que era esperada, mientras que el otro índice representará el número de etiqueta obtenida. El valor asignado de la matriz para ambos índices, sera el numero de casos donde tal situación ocurrió (por ejemplo, si el elemento $M_{ij} = 2$, significa que se identificó dos veces a la etiqueta i cuando la correcta era j).

Para realizar esta experimentación, se usó el dataset de 410 imágenes otorgadas por la cátedra. Para que cada imagen pudiera ser individualmente clasificada, se utilizó el algoritmo KFold de validación cruzada, de a diez pliegues. De esta manéra, por cada conjunto de testeo, se usaba la mayor cantidad de imágenes para entrenar el predictor, siendo estas seleccionadas con distribución bien proporcionada de muestras por clase distinta (como hemos explicado previamente). Así entonces, al realizar las iteraciones del KFold, cada conjunto disjunto de testeo sería evaluado, hasta terminar habiendo probado el conjunto de 410 imágenes en su totalidad.

Para poder evaluar cómo se ve afectada la matriz de confusión a medida que los parámetros k y α varían, se realizó primero un escenario donde el predictor tomó $\alpha = 80$ y $k = 20$.

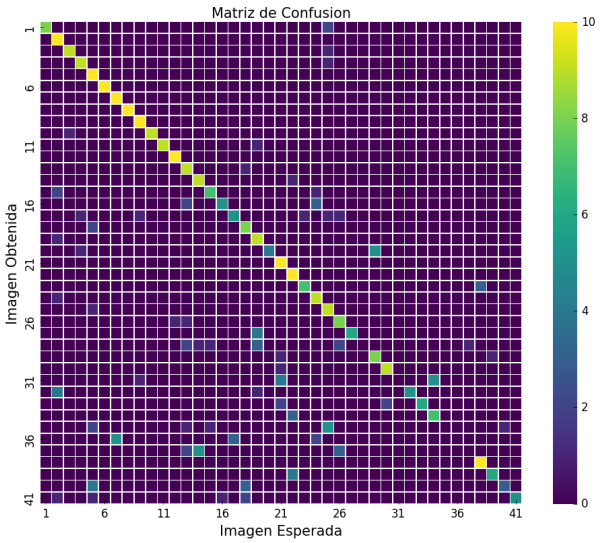


Figure 12: Matriz de confusión (80,20)

Para este escenario se puede ver analizando la diagonal, que para varios elementos distintos la cantidad de veces que fueron clasificados incorrectamente fue significativa, lo cual es correlativo con las experimentaciones anteriores, que identificaban esta combinación de α y k como una muy mala. Pero lo que aporta este análisis, es identificar qué imágenes en particular son las que más empeoran la *accuracy* del predictor. Por ejemplo, parece que las imágenes 34, 35 y 36 nunca se clasificaron correctamente, mientras que otras como las 2, 6 y 7 fueron mucho más frecuentes.

Una propiedad interesante a comentar acerca de estas matrices, es cómo utilizando la fila correspondiente a una imágen obtenida, se puede obtener la *precision*. Esto viene a ser ya que sumar la fila entera, resulta en todos los casos donde el predictor devolvió la etiqueta analizada, y por otro lado el elemento diagonal de dicha fila muestra los casos donde efectivamente la imagen era de esa clase. Dividiendo estos dos valores se obtiene entonces la *precision* al analizar de manera binaria la imagen i (i es un caso positivo, cualquier otro es un negativo). Así mismo, con las columnas se puede obtener el *recall*, ya que la sumatoria de los elementos de la columna da la totalidad de casos donde al predictor se le dió una etiqueta positiva, pero solo el elemento en la diagonal equivale a los casos donde el resultado fue también positivo.

Para medir el progreso a medida que cambian los parámetros del predictor, se tomó luego $\alpha = 70$ y $k = 10$.

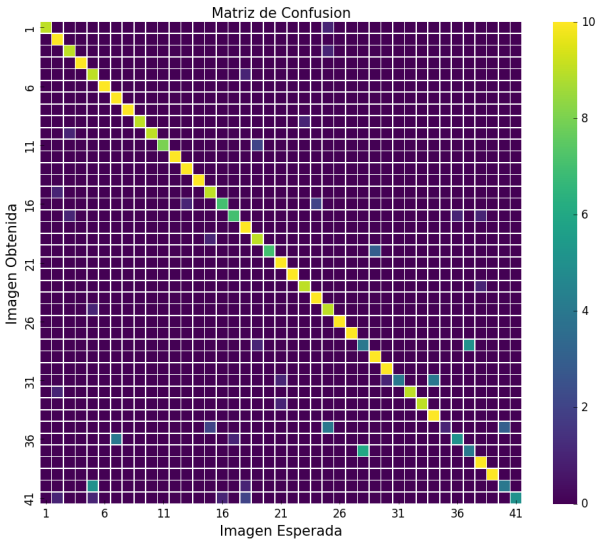


Figure 13: Matriz de Confusión (70,10)

A partir de este punto, ya se puede comenzar a apreciar cómo las predicciones empiezan a resultar en más valores positivos. En este caso, la imagen 35 es la única donde no se obtuvo ninguna predicción correcta, mientras que se redujeron las otras clases en las que previamente solo en la mitad de los casos se obtenían resultados positivos. Por último, analizamos con $\alpha = 65$ y $k = 5$, para los cuales según los análisis previos, ya se encuentran mucho más cercanos a los mejores valores para definir el predictor.

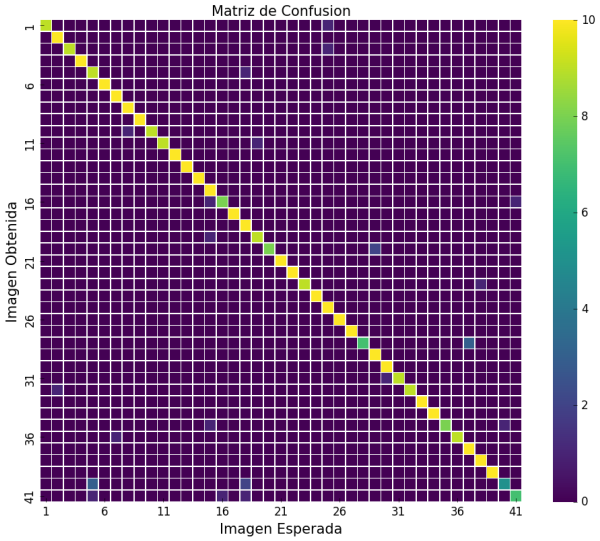


Figure 14: Matriz de Confusión (65,5)

Llegado este caso, es inevitable observar que ya no hay más clases que nunca fueron identificadas correctamente (como sí lo había en las matrices de confusión previas), mientras que la gran mayoría fue clasificada correctamente en todas sus muestras. En estas situaciones donde se puede ver que el gráfico consiste en una simple matriz diagonal, no hay mucha más información que aporte tal como antes donde se podía ver qué casos eran más prominentes a causar predicciones pobres. Sin embargo, hay que admitir que queda más lindo de esta última manera.

4.6 Algoritmo de Clasificación Alterno

En los experimentos anteriores resolvimos varios detalles sobre la clasificación con el algoritmo de KNN, y su variante

sumando PCA. Vimos cómo varía el comportamiento moviendo diferentes variables, y analizamos cómo actúa cada métrica en general.

Sin embargo nos quedamos con una duda: ¿es 'bueno' el algoritmo?. Podemos estar conformes o no con los resultados obtenidos: elegir utilizarlo o descartarlo dependiendo de cuáles sean nuestros requisitos mínimos. Sin embargo, nos aportaría un poco más de contexto el hecho de investigar otras opciones, y en este caso al menos compararlo con un algoritmo de selección alternativo pensado por nuestro grupo.

Dentro de esta sección vamos a desarrollar un breve experimento sobre nuestro algoritmo de clasificación llamado **K-Puntos**.

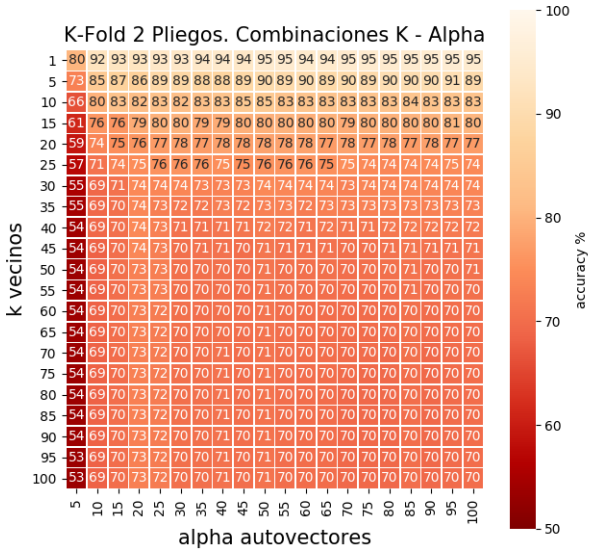
El algoritmo se basa en los siguientes pasos:

- Seleccionar las k imágenes más cercanas
- Asignar un puntaje a cada una por orden de aparición
  - La primer imagen seleccionada tiene k puntos
  - La segunda imagen seleccionada tiene $k-1$ puntos
  - ...
  - ...
  - La última imagen seleccionada tiene 1 punto
- Sumar el puntaje de cada etiqueta
- Aquella label que tiene más puntos gana

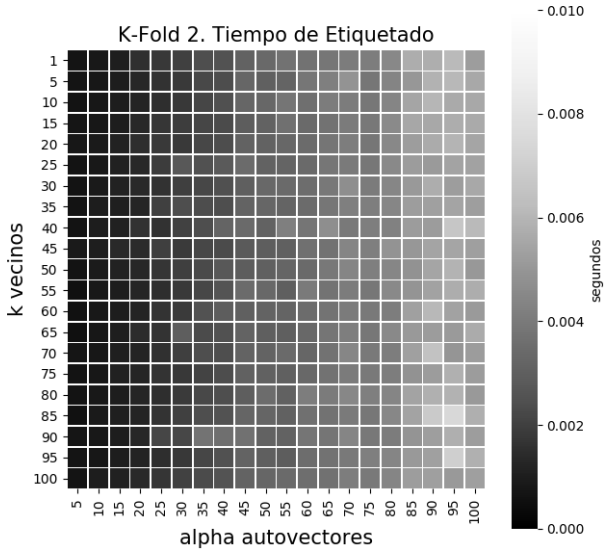
De esta manera planeamos poder revertir los casos en que correctamente se seleccionaron en lugares altos a las imágenes del label correcto, pero luego al ir aumentando el k, aparecieron mayor cantidad de una imagen incorrecta, haciendo que esta última sea la ganadora de la votación planteada en KNN

Para este algoritmo esperamos una mejor performance para k mayores, ya que la manera en que está planteado hace que a medida que se vayan aumentando la cantidad de vecinos a considerar estos tengan menos valor, por lo que si antes el aumento del k era un factor negativo, ahora esperamos que afecte en menor medida.

Por otro lado, esperamos un comportamiento parecido para casos pequeños. Así como KNN solía tener un porcentaje de aciertos bastante alto, creemos que $K-Puntos$ no afectará tanto el comportamiento en estas instancias.



(a) Figura 15: Accuracy



(b) Figura 15: Tiempo de ejecución

Como podemos observar obtuvimos unos resultados parecidos a lo que tenemos en KNN. Nuevamente vemos que la variable *alpha* a partir de $alpha = 40$ varía en muy baja medida con respecto al accuracy del sistema, pero marca gran diferencia en los tiempos.

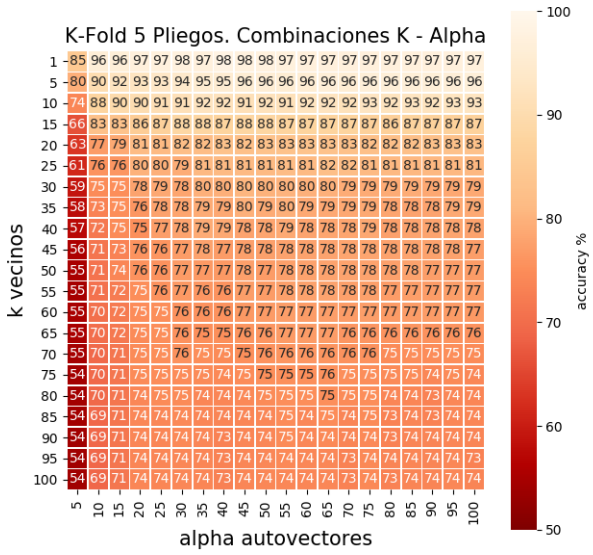
A su vez, podemos ver como cambia el comportamiento del algoritmo al rededor de la variable k, nuevamente vemos que el mayor salto entre el máximo y el mínimo de accuracy está relacionado con la cantidad de vecinos. Sin embargo, esta vez se muestra un sistema mucho más homogéneo, ya que a partir de $k = 5$ se genera una moda de accuracy de al rededor de 80%

A este fenómeno lo relacionamos especialmente con la principal característica de este algoritmos de selección, que se trata de que las primeras imágenes elegidas aporten mucho más peso que las últimas al etiquetado. Luego, aunque vayamos aumentando la cantidad de vecinos, es difícil que se cambie la votación inicial.

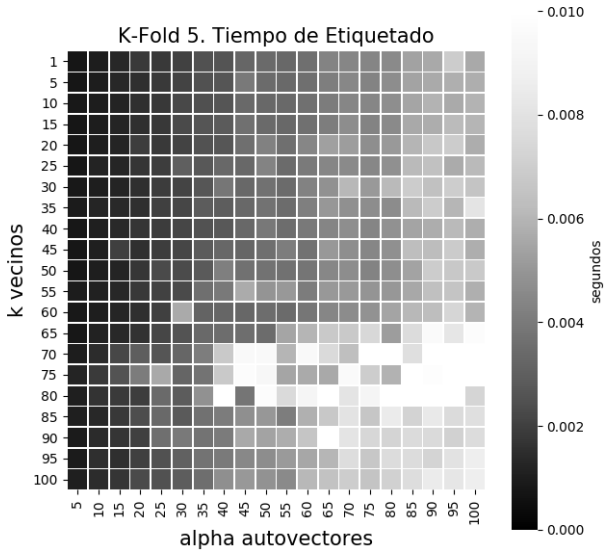
De este algoritmo podemos aportarle finalmente algo de contexto a KNN. En $K - Puntos$ vemos una accuracy mayor en general, aunque sin embargo en el caso óptimo, que es para $k = 1$, ambos muestran el mismo resultado, ya que ambas eligirian a la imagen más cercana como la respuesta.

Al final podemos concluir, por el momento, que por más que notamos valores muy buenos para KNN, actualmente logramos un algoritmo levemente mejor para la mayoria de las combinaciones (por lo menos para esta base), y con tiempos de computo dentro del mismo rango.

Sin embargo volveremos a realizar este experimento con $K = 5$, para ver si podemos observar el mismo fenómeno.



(a) Figura 16: Accuracy



(b) Figura 16: Tiempo de ejecución

Como esperamos al entrenar con más imágenes que la cantidad con las que se testea, obtenemos una mejor precisión promedio. Nuevamente vemos una mejora general utilizando $K - Puntos$ sobre KNN, y que nuevamente los casos óptimos al tomar a $k = 1$ son iguales.

4.7 Nueva Base de Datos

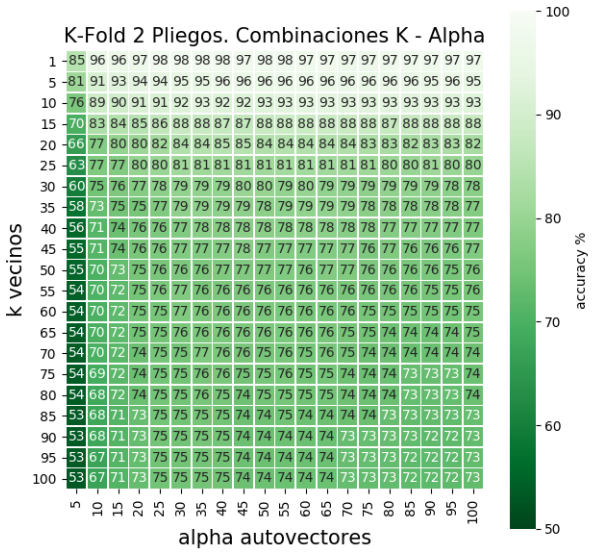
En esta sección planeamos volver a correr el experimento de accuracy, variando a k y *alpha* sobre una nueva base completamente diferente a la provista por la cátedra. De esta manera queremos ver si llegamos a encontrar algún comportamiento diferente de lo descripto hasta el momento, que se haya generado por producto de la base provista por la cátedra.

Obtuvimos la nueva Base de Imágenes de la página `http://www.cs.tau.ac.il/~wolf/ytfaces/`, el archivo *frame_images_DB.tar.gz*. Estas imágenes descargadas contienen varias fotos de diferentes famosos, etiquetadas y en formato jpg. Decidimos elegir de manera arbitraria a 41 famosos, tomar 10 fotos de cada uno/a, y transformarlas a formato pgm con el convertidor de *imagetrick* (las mismas son de $320x240$).

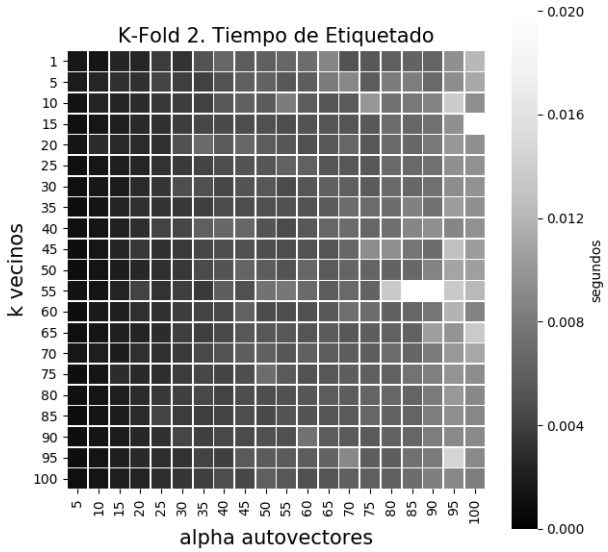
Esta nueva base además provee imágenes muy diferentes entre cada etiqueta, con lo que podemos chequear si existe algún nuevo comportamiento que no hayamos identificado con nuestra anterior base.

Para un comienzo vamos a realizar la experimentación con selección KNN, para K de *K-Fold* igual a 2 y 5 al igual que como realizamos los experimentos anteriores.

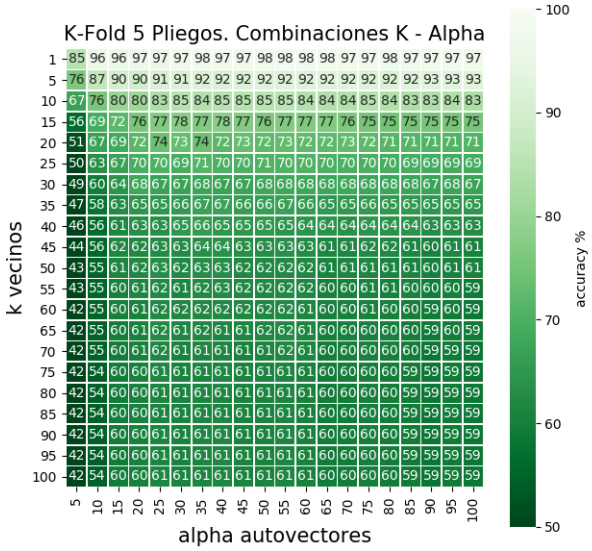
Los **resultados** fueron los siguientes:



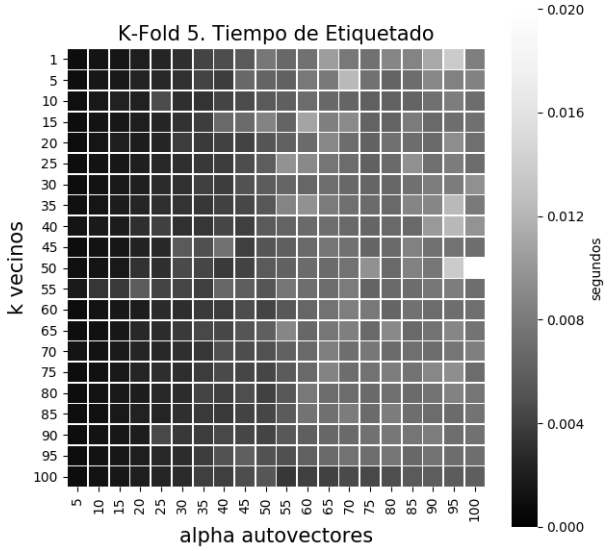
(a) Figura 17: Accuracy



(b) Figura 17: Tiempo de ejecución



(a) Figura 18: Accuracy



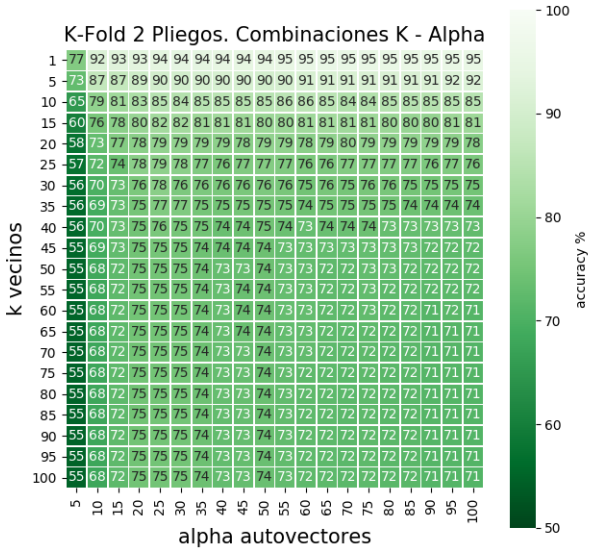
(b) Figura 18: Tiempo de ejecución

Con la nueva base notamos comportamientos muy similares a los antes provistos. En este caso, al tener una base con imágenes muy diferentes entre labels, el algoritmo suele confundirse menos entre dos imágenes, mostrando una **accuracy** bastante más alta.

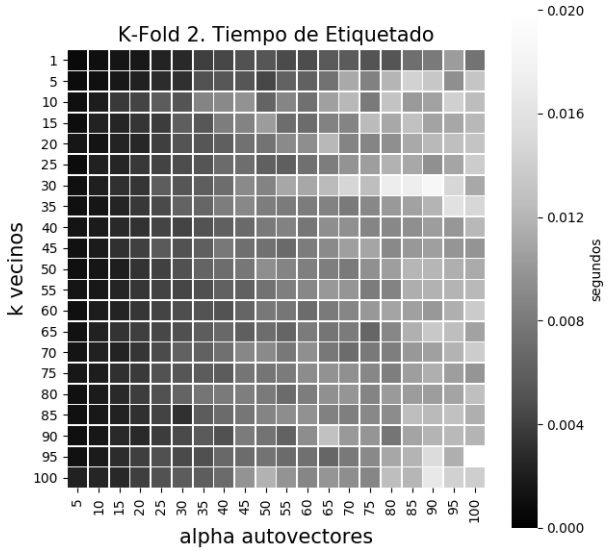
Sin embargo podemos notar un comportamiento extraño: para $K = 5$ obtenemos una **accuracy** general menor que para $K = 2$. Este comportamiento extraño se lo asignamos a la versatibilidad dentro de la base, y sacamos la conclusión de que por más que en general podamos sacar varias características del sistema, muchos elementos se ven afectados simplemente por la base de entrenamiento y test.

Los comportamientos los volvimos a realizar con el algoritmo $K - Puntos$, con el fin de comprobar si los resultados mantienen las mismas características anteriormente.

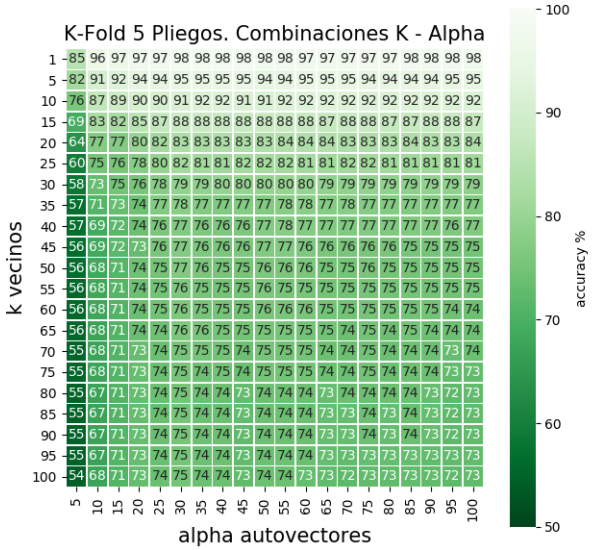
Por ello realizamos los siguientes gráficos:



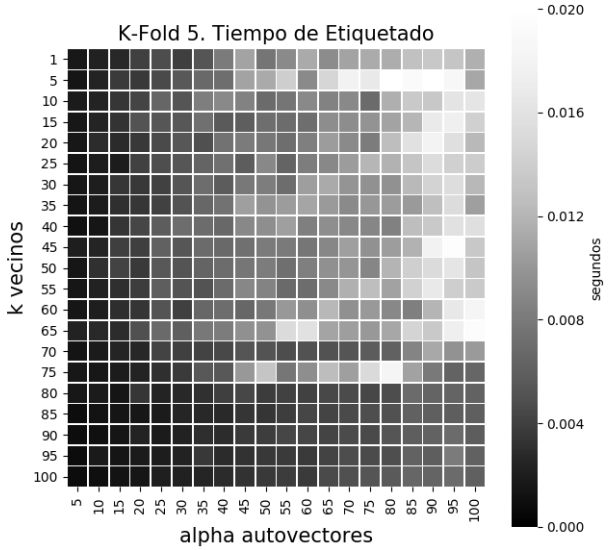
(a) Figura 17: Accuracy



(b) Figura 17: Tiempo de ejecución



(a) Figura 18: Accuracy



(b) Figura 18: Tiempo de ejecución

Nuevamente podemos ver que el algoritmo $K - Puntos$ muestra un mejor caso promedio para las combinaciones k - $alpha$, y que el caso óptimo sigue siendo el mismo, ya que analizan la misma imagen.

Las conclusiones de esta experimentación nos sirvieron para confirmar lo descripto en experimentos anteriores.

4.8 Experimentación Final

Ya habiendo realizado un análisis exhaustivo sobre cómo ciertos parámetros al ser fijados alteran la precisión de los algoritmos usados, decidimos fijar los parámetros óptimos y probar la eficiencia de los distintos métodos entre ellos. Con el fin de ver como la calidad de las predicciones aumenta en gran forma usando el análisis de componentes principales, compararemos los siguientes métodos con respecto a la base de datos provista por la cátedra:

1. KNN sin PCA
2. KNN con PCA
3. K-Puntos con PCA

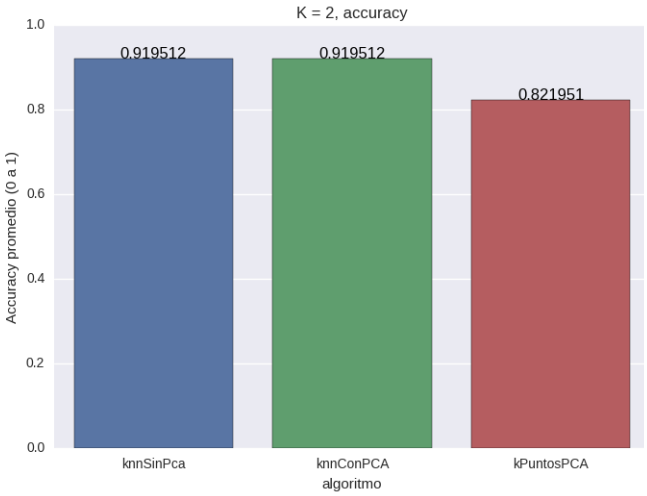
Los parámetros óptimos elegidos con los cuales realizaremos los experimentos sobre los tres métodos anteriores fueron:

1. $\alpha = 40$
2. $k = 5$ (kPuntos con PCA) y $k = 1$ (Knn con y sin PCA)

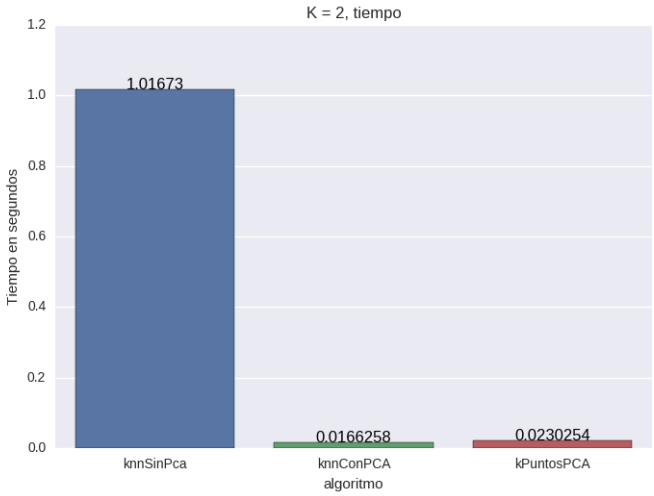
Estos fueron seleccionados como tales ya que tanto evaluando el accuracy como el puntaje F1, se llegó a las conclusiones de que para esa combinación de α y k los resultados eran los mejores obtenidos. Para KNN sin PCA, también se midió que el algoritmo retornaba la mayor cantidad de predicciones correctas para valores k entre 1 y 5. Con respecto a $k = 5$ en kPuntos con PCA, se eligió ese parámetro a pesar de que este tiene mejores resultados con $k = 1$, pues para este último valor el algoritmo sería idéntico a kNN.

Las métricas utilizadas para comparar todas estas metodologías fueron tanto el accuracy, como el tiempo de clasificación. Para presentar un panorama más amplio de resultados, ambos experimentos (tiempo de ejecución y accuracy) fueron realizados haciendo K-Fold cross-validation, fijando $K = 2$ y $K = 5$.

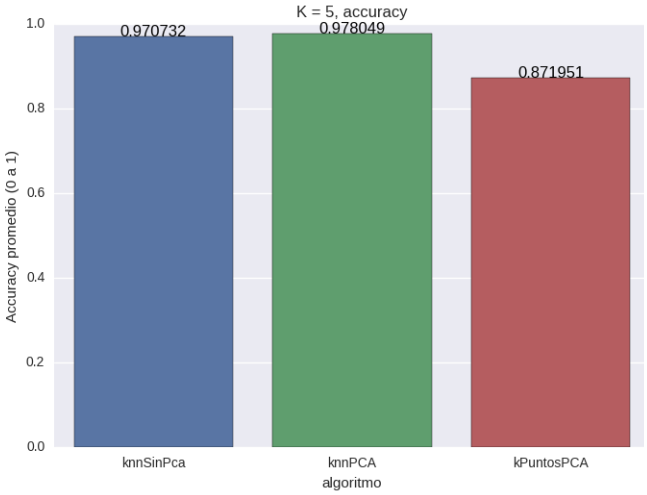
Resultados



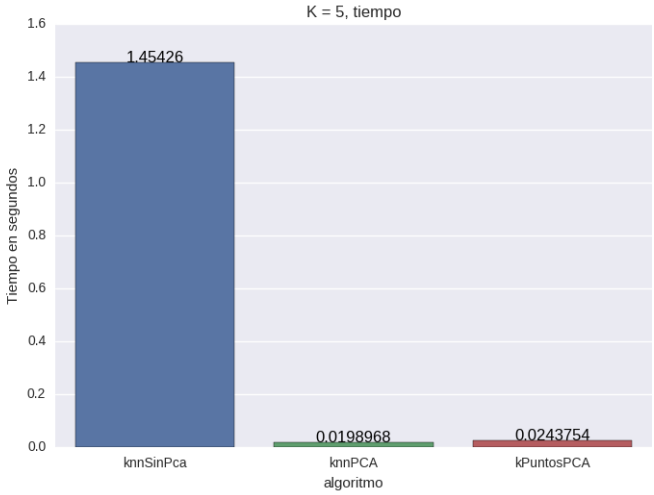
(a) Figura 19: Accuracy



(b) Figura 19: Tiempo de ejecución



(a) Figura 20: Accuracy



(b) Figura 20: Tiempo de ejecución

Discusión y conclusiones

Al observar las **Figuras 19 y 20** podemos contrastar lo que vimos a lo largo de toda nuestra experimentación con respecto a K: Tener más imágenes en el conjunto de entrenamiento va a incrementar las probabilidades de que la imagen sea identificada correctamente.

Con respecto a la comparación entre los tres métodos, se ve que la *accuracy* de kPuntos con PCA es más baja que la del resto. De todas formas, esto probablemente sea producto de haberlo corrido con $k = 5$, pues con $k = 1$ hubiese tenido resultados exactamente iguales que kNN con PCA.

Quedándonos con kNN con PCA vs kNN sin PCA, pareciera que por muy poco el primero supera al segundo. La diferencia es tan poca (y se ve sólo con $K = 5$) que estaríamos tentados a decir que estuvimos completamente equivocados al hablar, por ejemplo, de la `maldición de la dimensionalidad` y de la utilidad de aplicarle una transformación a la base de datos de entrada (que es lo que hace PCA). Sin embargo, creemos que de experimentar con otras bases de datos no tan bien distribuidas como esta y con más diferencia entre las distintas instancias de una misma cara, aplicarle el método Principal Component Analysis a los datos traerá un beneficio muy grande con respecto a la calidad de los resultados.

De todas formas, si tuviéramos que elegir uno de los métodos para utilizar siempre con nuestra base de datos, sin duda elegiríamos **Knn con PCA** con estos parámetros óptimos pues el tiempo de ejecución es muchísimo menor que sin realizar el preprocesamiento de las imágenes.

5 Conclusiones finales

El foco de este trabajo estuvo puesto en la experimentación científica de un método de reconocimiento de caras simplificado usando clasificación por vecinos más cercanos y reducción de la dimensionalidad, con el fin de obtener los mejores parámetros que dieran mejores resultados. A su vez, se plantearon de forma teórica conceptos necesarios para una mejor implementación del calificador. Se evaluaron los resultados en términos de accuracy, F1-Score, Matriz de confusión(Precision/Recall) y Tiempos de ejecución, para obtener un mayor marco representativo de cada algoritmo.

Sin duda la mayor parte de los experimentos se vieron atravesados por el análisis de la sección **4.2**, y mencionando nuestra implementación de utilizar los K-Folds uniformemente distribuidos. Pudimos constatar los resultados previamente formulados con las experimentaciones de *KNN* y *KNN + PCA*, donde a partir de cierto K el accuracy obtenido se mantenía estable.

De estos últimos dos experimentos pudimos determinar los mejores parámetros para la utilización de los mismos. Pudimos concluir en la fragilidad del parámetro k, ya que a partir de cierta cantidad de vecinos los resultados no muestran muchas diferencias. Un fenómeno similar se vio también con el parámetro *alpha*, para el cual notamos cómo en los extremos de los valores que puede tomar obtenemos malos resultados, ya sea por evaluar muy pocas columnas significativas de la matriz, lo que hace que las imágenes se parezcan mucho entre sí (caso alpha muy bajo), o bien caer en la *maldición de la dimensionalidad* si no lo utilizáramos, o tomáramos uno muy alto.

Por otro lado, hemos tomado una base de datos diferente a la de la cátedra y sin embargo obtuvimos resultados muy similares, lo que pareciera indicar que cambiar de base de datos no hace que se modifiquen los resultados. Sin embargo, en nuestra base alterna las instancias de cada etiqueta eran similares, por lo cual podríamos ver potenciales cambios de utilizar otra base en la cual las instancias de cada etiqueta no sean muy parecidas, o que las cantidad de imágenes de cada etiqueta no esté bien distribuida.

El método de *PCA* no resultó ser de mucha eficiencia para el problema que se intentó evitar. Los resultados de accuracy no variaron mucho pero sí se optimizó el tiempo de ejecución. Esto se puede deber a múltiples causas:

- Que no se haya modificado sustancialmente el accuracy obtenido pues los pixeles más significativos de las imágenes no son muy distintos entre si. Esto puede deberse en parte a que son todas en blanco y negro y no logran captar una gama amplia de variables al momento de compararlas.
- Otra posibilidad es también que el problema de la *maldición de la dimensionalidad* no se puede obtener en la base de datos con las que trabajamos. Puede deberse al tamaño de las mismas o a la cantidad de ellas.

Finalmente con respecto a las otras métricas experimentadas (F1-score, precisión, recall) se observó que los resultados obtenidos fueron coincidentes con el accuracy experimentado anteriormente. El calificador posee un buen rendimiento en todas sus métricas con los parámetros óptimos conseguidos.

Como conclusión final, sentimos que el algoritmo de *kNN* como calificador de imágenes no es del todo efectivo y está muy condicionado por la base con la cual se esta testeando. Algoritmos como el alterno propuesto (*K-Puntos*) poseen una mejor performance sin necesidad de condicionar la base inicial. Tratar a las imágenes vecinas con el mismo valor al momento de definir la etiqueta resultado no es una buena opción teniendo en cuenta que a más vecinos el algoritmo empieza a fallar. Pese a esto, no podríamos definir al calificador como **malo** dado que en promedio siempre hubo más probabilidades de acierto en cualquier métrica ($> 50\%$).

6 Apéndice A

Métodos Numéricos
1^o Cuatrimestre 2018
Trabajo Práctico 2

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Reconocimiento de caras

Introducción

Como plan estratégico para el desarrollo del país, las autoridades nacionales están a días de inaugurar el prometedor *soja valley* que será el pilar económico de los años venideros. Dado que el rimbombante emprendimiento albergará a las industrias más importantes para el país, se necesita un sistema de reconocimiento de trabajadores que ingresen a las instalaciones dado que se estará manejando información confidencial a diario.

Si bien es muy factible que existan mejores formas y más económicas para reconocer diariamente a un trabajador, las autoridades están muy interesadas en desarrollar un sistema biométrico basado en reconocimiento facial.

En este contexto, las autoridades han contactado al Departamento de Computación para el desarrollo del sistema de reconocimiento facial que se utilizará en el flamante *soja valley*. Como punto de partida para la realización de un prototipo, nos han provisto de una base de datos preliminar para poder realizar una prueba de concepto.

El foco de este trabajo está puesto en la experimentación científica de un método de reconocimiento caras simplificado usando clasificación por vecinos más cercanos y reducción de la dimensionalidad.

Metodología

Se quiere desarrollar un algoritmo de clasificación supervisada el cual se deberá *entrenar* con una base de caras **conocidas** que luego nos servirá para reconocer otras instancias de esas caras no presentes en la base de datos de entrenamiento.

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 y del mismo tamaño. Cada una de estas imágenes sabemos a qué persona corresponde.

Reconocimiento de caras

El objetivo del reconocimiento de caras consiste en utilizar la información de la base de datos para, dada una nueva imagen de una cara sin etiquetar, determinar a quién corresponde. En este trabajo práctico nos vamos a basar en el trabajo de Turk and Pentland [3] en el cual se plantea un esquema muy simple de reconocimiento de caras.

Una primera aproximación es utilizar el conocido algoritmo de k *vecinos más cercanos* o kNN [2], por su sigla en inglés. En su versión más simple, este algoritmo considera a cada objeto de la base de entrenamiento como un punto en el espacio euclídeo m-dimensional, para el cual se conoce a qué clase corresponde (en nuestro caso, qué persona es) para luego, dado un nuevo objeto, asociarle la clase del o los puntos más cercanos de la base de datos.

Procedimiento de k vecinos más cercanos

- Se define una base de datos de entrenamiento como el conjunto $\mathcal{D} = \{x_i \,:\, i = 1, \ldots, n\}$.
- Luego, se define m como el número total de píxeles de la imagen i-ésima almacenada por filas y representada como un vector $x_i \in \mathbb{R}^m$.
- De esta forma, dada una nueva imagen $x \in \mathbb{R}^m$, talque $x \notin \mathcal{D}$, para clasificarla simplemente se busca el subconjunto de los k vectores $\{x_i\} \subseteq \mathcal{D}$ más cercanos a x, y se le asigna la clase que posea el mayor número de repeticiones dentro de ese subconjunto, es decir, la moda.

El algoritmo del vecino más cercano es muy sensible a la dimensión de los objetos y a la variación de la intensidad de las imágenes. Es por eso, que las imágenes dentro de la base de datos $\mathcal{D}$ se suelen *preprocesar* para lidiar con estos problemas.

Teniendo en cuenta esto, una alternativa interesante de preprocesamiento es buscar reducir la cantidad de dimensiones de las muestras para trabajar con una cantidad de variables más acotada y, simultáneamente, buscando que las nuevas variables tengan información representativa para clasificar los objetos de la base de entrada.

En esta dirección, consideraremos el método de reducción de dimensionalidad *Análisis de componentes principales* o PCA (por su sigla en inglés) dejando de la lado los procesamientos de imágenes que se puedan realizar previamente o alternativamente a aplicar PCA.

Análisis de componentes principales

El método de análisis de componentes principales o PCA consiste en lo siguiente.

Sea $\mu = (x_1 + \ldots + x_n)/n$ el promedio de las imágenes $\mathcal{D} = \{x_i : i = 1, \ldots, n\}$ tal que $x_i \in \mathbb{R}^m$. Definimos $X \in \mathbb{R}^{n \times m}$ como la matriz que contiene en la i-ésima fila al vector $(x_i - \mu)^t/\sqrt{n-1}$. La matriz de covarianza de la muestra X se define como $M = X^t X$.

Siendo v_j el autovector de M asociado al j-ésimo autovalor, al ser ordenados por su valor absoluto, definimos para $i = 1, \ldots, n$ la *transformación característica* de x_i como el vector $\mathbf{tc}(x_i) = (v_1\, x_i, v_2\, x_i, \ldots, v_\alpha\, x_i)^t \in \mathbb{R}^\alpha$, 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 idea es que $\mathbf{tc}(x_i)$ resuma la información más relevante de la imagen, descartando los dimensiones menos significativas.

Imágenes de gran tamaño

La factibilidad de aplicar PCA es particularmente sensible al tamaño de las imágenes de la base de datos. Por ejemplo, al considerar imágenes en escala de grises de 100×100 píxeles implica trabajar con matrices de tamaño 10000×10000. Una alternativa es reducir el tamaño de las imágenes, por ejemplo, mediante un submuestreo (eliminación intercalada de filas y columnas de la imagen).

Sin embargo, es posible superar esta dificultad en los casos donde el número de muestras es menor que el número de variables, ya que es posible encontrar una relación entre los

autovalores y autovectores de la matriz de covarianza $M = X^t * X$ y la matriz $\hat{M} = X * X^t$.

Clasificación con kNN y PCA

El método PCA previamente presentado sirve para realizar una transformación de los datos de entrada a otra base y así trabajar en otro espacio con mejores propiedades que el original. Por lo tanto, el proceso completo de clasificación se puede resumir como:

Dada una nueva imagen x de una cara, se calcula $\mathbf{tc}(x)$, y se compara con $\mathbf{tc}(x_i)$, $\forall x_i \in \mathcal{D}$, para luego clasificar mediante kNN.

Validación cruzada

Finalmente, nos concentramos en la evaluación de los métodos y en la correcta elección de sus parámetros.

Dado que necesitamos conocer previamente a qué persona corresponde una imagen para poder estimar la correctitud de la clasificación, una alternativa es particionar la base de entrenamiento en dos, utilizando una parte de ella en forma completa para el entrenamiento y la restante como test, pudiendo así corroborar la clasificación realizada, al contar con el etiquetado del entrenamiento. Sin embargo, realizar toda la experimentación sobre una única partición de la base podría resultar en una incorrecta estimación de parámetros, dando lugar al conocido problema de *overfitting*.

Por lo tanto, se estudiará la técnica de *cross validation*[2], en particular el *K-fold cross validation*[1], para realizar una estimación de los parámetros de los métodos que resulte estadísticamente más robusta.

Enunciado

Se pide implementar un programa en `C` o `C++` que lea desde archivos las imágenes de entrenamiento correspondientes a distintas caras y que, utilizando los métodos descriptos en la sección anterior, dada una nueva imagen de una cara determine a qué persona pertenece.

Para ello, el programa **deberá** implementar el algoritmo de kNN así como también la reducción de dimensión utilizando PCA.

Con el objetivo de obtener la transformaciones características de cada método, **se deberá** implementar el método de la potencia con deflación para la estimación de autovalores/autovectores de la matriz de covarianza en el caso de PCA.

Se recomienda realizar tests para verificar la implementación del método en casos donde los autovalores y autovectores sean conocidos se antemano. También, puede resultar de utilidad comparar con los datos provistos por la cátedra, utilizando Matlab/Octave o alguna librería de cálculo numérico.

[1] No confundir el K de K-fold con el k de kNN, ambos son parámetros de los métodos respectivos que no están relacionados necesariamente

Se pide realizar un estudio experimental de los métodos propuestos sobre una base de entrenamiento utilizando la técnica *K-fold cross validation* mencionada anteriormente, con el objetivo de analizar el poder de clasificación y encontrar los mejores parámetros de los métodos.

Se pide desarrollar una herramienta alternativa que permita trabajar bajo ciertas condiciones con imágenes de tamaño mediano/grande y probar lo siguiente:

- Dada una matriz de covarianza $M = X^t * X$, encontrar una relación entre sus autovalores y sus autovectores con los de la matriz $\hat{M} = X * X^t$.

Se **deberá** trabajar al menos con la siguiente base de datos caras:
`http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`.

Experimentación

Para guiar la experimentación, se detallan los siguientes lineamientos y preguntas:

- Analizar la calidad de los resultados obtenidos al combinar kNN con y sin PCA, para un rango amplio de combinaciones de valores de k y α. Llamamos k a la cantidad de vecinos a considerar en el algoritmo kNN y α a la cantidad de componentes principales a tomar.
- Analizar la calidad de los resultados obtenidos al combinar kNN con PCA, para un rango amplio de cantidades de imágenes de entrenamiento. Utilizar desde muy pocas imágenes de entrenamiento hasta todas las disponibles para identificar en que situación se comporta mejor cada uno de los métodos.
- ¿Cómo se relaciona k con el tamaño del conjunto de entrenamiento? Pensar el valor máximo y mínimo que puede tomar k y qué sentido tendrían los valores.

También, **se debe** considerar en los análisis anteriores el tiempo de ejecución.

La calidad de los resultados obtenidos será analizada mediante diferentes métricas:

1. Accuracy
2. Precision/recall
3. F1-Score

En particular, la métrica más importante que **debe** reportarse en los experimentos es la tasa de efectividad lograda o *accuracy*, es decir, la cantidad caras correctamente clasificadas respecto a la cantidad total de casos analizados. También, se **debe** utilizar al menos otra de las métricas mencionadas, aunque no necesariamente para todos los experimentos realizados.

- Realizar los experimentos de los ítems anteriores para valores distintos de K del método K-fold[2], donde K a la cantidad de particiones consideradas para el cross-validation.

[2]Para esta tarea en particular, se recomienda leer la rutina `cvpartition` provista por Octave/MATLAB.

- Justificar el por qué de la elección de los mismos. ¿Cuál sería su valor máximo?
- ¿En qué situaciones es más conveniente utilizar K-fold con respecto a no utilizarlo?
- ¿Cómo afecta el tamaño del conjunto de entrenamiento?

- En base a los resultados obtenidos para ambos métodos, seleccionar aquella combinación de parámetros que se considere la mejor alternativa, con su correspondiente justificación, compararlas entre sí y sugerir un método para su utilización en la práctica.

En todos los casos es **obligatorio** fundamentar los experimentos planteados, proveer los archivos e información necesaria para replicarlos, presentar los resultados de forma conveniente y clara, y analizar los mismos con el nivel de detalle apropiado. En caso de ser necesario, es posible también generar instancias artificiales con el fin de ejemplificar y mostrar un comportamiento determinado.

Puntos opcionales (no obligatorios)

- Mostrar que si tenemos la descomposición $M = U\Sigma V^t$, V es la misma matriz que obtenemos al diagonalizar la matriz de covarianzas.
- Realizar experimentos utilizando otras imágenes de caras tomadas por el grupo. Tener en cuenta lo mencionado sobre el tamaño de las matrices a procesar con PCA. Reportar resultados y dificultades encontradas.
- Proponer y/o implementar alguna mejora al algoritmo de kNN. Por ejemplo, no considerar votación y utilizar la cercanía a la media de de cada clase como criterio de clasificación.
- Implementar y experimentar un métodos de detección de caras para encontrar si una imagen contiene o no una cara. Proponer un valor de confianza para la respuesta de la detección. Es decir, se quiere intentar responder la pregunta de si el sistema es capaz no solo de reconocer caras con los que fue entrenado, sino si es posible discernir entre una imagen de una cara que no se encontraba en la base de entrenamiento y una imagen con un objeto o contenido que no sea una cara.
- Aplicar técnicas de procesamiento de imágenes a las imágenes de caras previo a la clasificación[1]. Analizar como impacta en la clasificación la alteración de la intensidad de los píxeles, el ruido introducido en las imágenes y la variación de la ubicación y posición de las personas.

Formato de entrada/salida

El ejecutable producido por el código fuente entregado deberá contar con las funcionalidades pedidas en este apartado. El mismo deberá tomar al menos tres parámetros por línea de comando con la siguiente convención:

```
$ ./tp2 -m <method> -i <train_set> -q <test_set> -o <classif>
```

donde:

- `<method>` el método a ejecutar con posibilidad de extensión (0: kNN, 1: PCA + kNN, ... etc)
- `<train_set>` será el nombre del archivo de entrada con los datos de entrenamiento.
- `<test_set>` será el nombre del archivo con los datos de test a clasificar
- `<classif>` el nombre del archivo de salida con la clasificación de los datos de test de `<test_set>`

Todos los archivos de entrada/salida deberán estar en `.csv` y siguiendo el formato siguiente:

```
<archivo1>, id1,
 ...

<archivoN>, idN,
```

donde `idX` es un entero positivo entre 1 y la cantidad de personas en el dataset.

Un ejemplo de invocación con los datos provistos por la cátedra sería el siguiente:

```
$ ./tp2 -m 1 -i train.csv -q test.csv -o result.csv
```

Además, el programa deberá imprimir por consola un archivo, cuyo formato queda a criterio del grupo, indicando la tasa de reconocimiento obtenida para cada conjunto de test y los parámetros utilizados para los métodos.

Nota: cada grupo tendrá la libertad de extender las funcionalidades provistas por su ejecutable. En particular, puede ser de utilidad alguna variante de toma de parámetros que permita entrenar con un porcentaje de la base de datos de entrenamiento y testear con el resto (ver archivos provistos por la cátedra). Además, puede ser conveniente separar la fase de entrenamiento de la de testeo/consulta para agilizar los cálculos.

Fecha de entrega

- Formato Electrónico: Viernes 25 de Mayo de 2018 hasta las 23:59 hs, enviando el trabajo (informe + código) a la dirección `metnum.lab@gmail.com`. El subject del email debe comenzar con el texto `[TP2]` seguido de la lista de apellidos de los integrantes del grupo separados por punto y coma `;`.
  Se ruega no sobrepasar el máximo permitido de archivos adjuntos de 20MB. Tener en cuenta al realizar la entrega de no ajuntar bases de datos disponibles en la web, resultados duplicados o archivos de backup.
- Formato físico: Lunes 28 de Mayo de 2018 a las 18 hs. en la clase de laboratorio.
- Pautas de laboratorio: `http://www-2.dc.uba.ar/materias/metnum/homepage.html`

Importante: El horario es estricto. Los correos recibidos después de la hora indicada serán considerados re-entrega.

References

[1] Harold Hotelling. "Analysis of a complex of statistical variables into principal components". In: *Journal of Educational Psychology* 24 (1933), pp. 417–441.

[2] Richard Von Mises and H Pollaczek-Geiringer. "Praktische Verfahren der Gleichungsauflösung". In: *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik* 9 (1929), pp. 152–164.