DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

Reconocimiento de caras

Métodos Numéricos

Primer Cuatrimestre de 2018

Integrante	LU	Correo electrónico
Serio, Franco Agustìn	215/15	francoagustinserio@gmail.com
Ruiz Ramirez, Emanuel David	343/15	ema.capo2009@hotmail.com
Goldstein, Brian Uriel	27/14	brai.goldstein@gmail.com
Fadel, Uriel	104/14	uriel.fadel@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. Introducción Teórica

Palabras claves: PCA - knn - K-fold Cross Validation - Método de la Potencia

Resumen: Se propone un método de reconocimiento de caras, el cual comienza con una base de datos de caras de personas y que dada una nueva cara determine a qué persona de la base de datos corresponde. Este método hará uso de que por cada cara de la base de datos se extraerá la información más relevante y también saber cuales son las caras más parecidas a la nueva cara. En este trabajo se analizará cuánta información extraer por cara y cuántas de las más parecidas a la nueva cara hacen falta tomar, para que el método funcione lo mejor posible.

1.1. K-fold Cross Validation

La validación K-fold es una técnica para separar las muestras que se utilizarán para test y para training y es el primer paso en el TP. En nuestro caso, el código para dicha cuenta está en el main.cpp de la carpeta K-Fold. Siempre requerimos que el K en cuestión sea divisor de la cantidad de imágenes que tenemos por sujeto ya que éstas se dividirán en K particiones de igual tamaño. Una de esas particiones se usará para test y las restantes K - 1 para training en las K iteraciones del algoritmo, haciendo rotar las muestras. Ésto tiene la ventaja que todas las observaciones se usen tanto para test como para training.

1.2. Método de la Potencia

Sea A $\in$ R^{nxn} simétrica, con autovalores $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$, con base ortonormal de autovectores $\{v_1, v_2, ..., v_n\}$, se elije un y_0 tal que $v_1^t y_0 \neq 0$ y con $||.||$ un norma arbitraria. Entonces se puede obtener λ_1 y v_1.

Si se define la sucesión $y_{j+1} = \frac{A.y_j}{||A.y_j||}$ con $j = 0, 1, ...$ Y se quisiera calcular $y_{j'}$ donde j' es entero entonces $y_{j'}$ se lo puede considerar como v_1. Por último, para obtener λ_1 se hace la siguiente cuenta $\lambda_1 = \frac{y_{j'}^t.A.y_{j'}}{y_{j'}^t.y_{j'}}$

1.3. Deflación

Sea A $\in$ R^{nxn}, con autovalores $|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$, con base ortonormal de autovectores $\{v_1, v_2, ..., v_n\}$. Entonces se puede armar $A - v_1.v_1^t.\lambda_1 =$ B $\in$ R^{nxn} la cual cumple con tener como autovalores a 0 y $|\lambda_2| > ... > |\lambda_n|$ y como autovectores los mismos que A.

Combinando el Método de la Potencia con deflación se pueden hallar todos los autovalores y autovectores de una matriz. Siempre y cuando cumpliendo las hipótesis de ambas propiedades.

1.4. PCA

Dado un conjunto de n vectores $x_i \in$ R^m, un $\alpha \in \{1, ..., m\}$, definimos μ como el promedio de los vectores, X una matriz donde cada una de sus filas va a ser igual a $(x_i - \mu)/\sqrt{n-1}$ y M=X^t.X. Luego obtenemos los autovectores v_j de M asociados al autovalor λ_j. Para poder calcular estos v_j y λ_j se hace uso del método de la potencia combinado con deflación. Con estos autovalores ordenados de mayor a menor se puede definir la transformación característica de x$_i$ como tc(x$_i$)=$(v_1^t x_i, v_2^t x_i, ..., v_\alpha^t x_i) \in R^\alpha$. Por lo tanto, se obtienen los tc(x_i) los cuales forman un conjunto de n vectores que tienen menos dimensiones que los x_i.

1.5. Knn

El algoritmo Knn (por K-nearest-neighbours) es un clasificador de clases y es el último paso en nuestro TP. En este caso la transformación característica de cada imagen de test de nuestro TP, es comparada con la transformación característica de cada imagen de la muestra de train, tomando la distancia euclidea, es decir, $\sqrt{\sum_{r=1}^{\alpha}(tc(x^*)[r] - tc(x_i)[r])^2}$, donde $tc(x^*)$ es la transformación característica de la imagen de test, $tc(x_i)$ la transformación característica de la i-ésima imagen de training y α la cantidad de componentes principales que queremos conservar. Con esto, podemos realizar una votación de clases de acuerdo al k del algoritmo de Knn. Las distancias obtenidas se ordenan de menor a mayor y se toman los k primeros resultados. La clase más votada en ésta última muestra es la ganadora y con eso damos la estimación de la imagen de test ya que tenemos los labels o etiquetas de cada una de las clases de training.

1.6. Presicion/recall

En el contexto del clasificador de caras el cual es un clasificador multiclase, donde cada clase es un sujeto. Si se tienen las clases i=1...N, se calcula para cada una: tp_i, fp_i , fn_i y tn_i.

- tp_i:(true positive) cantidad de caras que se clasificaron como de la clase i y estas eran de esa clase.
- fp_i:(false positive) cantidad de caras que se clasificaron como de la clase i pero estas no eran de esa clase.
- fn_i:(false negative) cantidad de caras que el clasificador indico que eran de una clase distinta a la i cuando estas eran de la clase i.
- tn_i:(true negative) cantidad de caras que el clasificador indico que eran de una clase distinta a la i y estas eran de una clase distinta a la i.

Entonces se puede definir por cada clase $presicion_i = \frac{tp_i}{tp_i + fp_i}$ y $recall_i = \frac{tp_i}{tp_i + fn_i}$

La **precisión** del clasificador multiclase, se define como el promedio de las precisión de cada una de las clases y el **recall** de este clasificador como el promedio de los recall de cada una de las clases.

1.7. Accuracy

En el contexto del clasificador de caras, **Accuracy** se lo define como cantidad de caras bien clasificadas sobre el total de caras por clasificar. Esta métrica tiene la ventaja de que es fácil de entender. Pero como contra, que si este valor es alto y se tiene un sujeto que esta asociado a muchas caras, eso nos diría que el clasificador multiclase anda bien solo para esa clase mientras que para el resto no. Es decir, que termina ocultado algunos detalles de la clasificación.

1.8. f1-score

En el contexto del clasificador multiclase, f1-score es la media armónica de precision y recall de este tipo de clasificador. Además pondera precision y recall de forma equitativa y se la puede calcular como f1-score $= \frac{2.(Recall.Precision)}{(Recall + Precision)}$. A pesar de que no parezca de fácil de comprensión, esta métrica tiene la ventaja indicar que tan bien funciona este clasificador cuando las clases no están distribuidas de la misma forma.

2. Desarrollo

Sea n=cantidad de imágenes en la base de entrenamiento y m=cantidad de píxeles de una imagen.
X $\in \mathrm{R}^{nxm}$, X^t.X $\in \mathrm{R}^{mxm}$ y X.X^t $\in \mathrm{R}^{nxn}$
Para el cálculo de autovalores:
X = $U\Sigma V^t$ la descomposición svd con U y V ortogonales.
X.X^t = $U\Sigma V^t.(U\Sigma V^t)^t = U\Sigma V^t.V\Sigma^t U^t = U.\Sigma.\Sigma^t.U^t$
X^t.X = $(U\Sigma V^t)^t.U\Sigma V^t = V\Sigma^t U^t.U\Sigma V^t = V\Sigma^t.\Sigma V^t$
Como $\Sigma.\Sigma^t$ tiene en la diagonal (los cuadrados de valores singulares positivos de X) σ_i^2 con $1 \leq i \leq rg(X)$ y 0, estos son autovalores de X.X^t. Y $\Sigma^t.\Sigma$ tiene en la diagonal (los cuadrados de valores singulares positivos de X^t) σ_i^2 con $1 \leq i \leq rg(X)$ y 0, estos son autovalores de X^t.X.
Entonces X.X^t y X^t.X tienen los mismos autovalores.

Para el cálculo de autovectores:
Las columnas de V (v_i) son los autovectores de X^t.X.
Las columnas de U (u_i)son los autovectores de X.X^t.
$\mathrm{X}^t = V\Sigma^t U^t \Leftrightarrow \mathrm{X}^t.\ \mathrm{U} = V\Sigma^t$
Se puede deducir que $\mathrm{X}^t.\mathrm{u}_i = \sigma_i.\mathrm{v}_i \Leftrightarrow \frac{X^t.u_i}{\sigma_i} = \mathrm{v}_i$ con $1 \leq i \leq g(X)$
Teniendo los autovectores de X^t.X, entonces podemos calcular los autovectores de X^t.X.

2.1. Método de la potencia y deflación

Los métodos de deflación y método de la potencia, se utilizan en el TP para diagonalizar y sacar los autovectores y autovalores de $M = X^t * X$

Para el TP se realizó un algoritmo recursivo para el método de la potencia y deflación. Se tiene una clase, llamada PCA cuyos únicos elementos privados son los autovectores y autovalores. El constructor de ésta clase llama al método de la potencia con $M = X^t * X$, el número de iteraciones para que el método converja, el α que es la cantidad de componentes que vamos a querer guardar y los autovectores y autovalores que son dos vectores donde se irán guardando los autovectores y autovalores respectivamente.
Después de sacar el primer par de autovector y autovalor, con deflación modificamos (achicamos) M y la volvemos a tomar para volver a correr el método de la potencia. Así hasta obtener α autovectores y autovalores. El método se encarga de sacar autovectores de la matriz, multiplicando ésta por el autovector recursivamente la cantidad de veces que indique el número de iteraciones. Para los autovalores simplemente remplazamos el autovector obtenido en la ecuación $\lambda = \frac{v^T M^T v}{v^T v} = \frac{(Mv)^T v}{v^T v}$. Siempre para la primer pasada se usa un vector aleatorio. Todos los vectores son normalizados.

2.2. Detalles de implementación

Para representar los números decimales se optó por tipo $double$. Porque se pueden representar muchos valores.

Para representar matrices se decidió usar $vector < vector < double >>$. Porque con esta representación, las implementaciones de las operaciones de matrices son fáciles de hacer.

3. Experimentación

3.1. Experimento 1:

En este experimento buscamos evaluar el método de la potencia como método para encontrar autovalores de una matriz. Para esto el experimento que propusimos fue observar la diferencia entre los autovalores calculados por nuestro algoritmo y los autovalores computados por la librería numpy y ver como varia esta diferencia a medida que se incrementa el tamaño de la matriz y la cantidad de iteraciones.
El objetivo del experimento sera determinar un valor (mínimo) para el parámetro ”cantIter” (la cantidad de iteraciones del método de la potencia) de modo que sea lo suficientemente grande como para calcular con poco error los autovalores y autovectores.

3.1.1. Experimento 1-A

Para responder esto generamos matrices aleatorias de tamaño incremental. Computamos sus autovalores utilizando una librería de álgebra lineal (a estos los llamamos ”autovalores optimos”) y los comparamos con los autovalores que calcula nuestro algoritmo para una determinada cantidad de iteraciones (a estos los llamamos ”autovalores calculados”). Este Proceso lo repetimos variando el tamaño de la matriz (siempre es cuadrada y variamos su ancho y alto) y variando también la cantidad de iteraciones que usa nuestro algoritmo para computar autovalores.

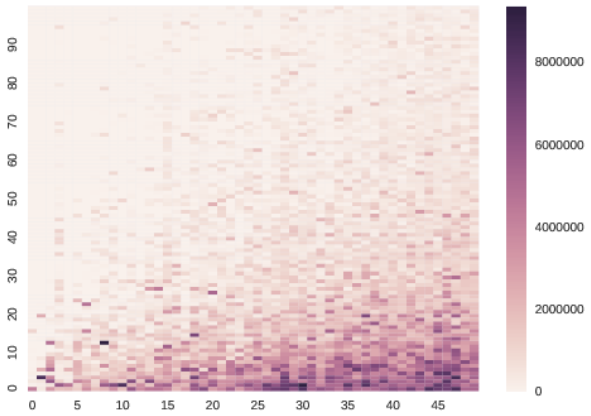


Figura 1: Dimensión de matriz (cuadrada) en el eje de abscisas. Cantidad de iteraciones del método de la potencia en las ordenadas. Diferencia entre ”autovalores calculados” y ”autovalores óptimos” en el coloreo (mas intenso significa mas diferencia).

3.1.2. Observaciones:

Se puede ver como las primeras iteraciones influyen mucho mas ajustando los autovalores. Se observa también como si bien hay una relación entre el tamaño de la matriz y la cantidad de iteraciones necesarias, no llega a ser lineal esta relación (o es a lo sumo lineal). Esto puede observarse en que si miramos las posiciones (x, x) para cualquier x encontramos que el coloreo es relativamente claro. Esto podría darnos la pauta de que tomar ”cantIter” proporcional al ancho/alto de la matriz puede ser una buena idea.

3.1.3. Experimento 1-B

El el inciso A del experimento 1 nos parece sugerir tomar un ”CantIter”(cantidad de iteraciones del método de la potencia) proporcional al ancho/alto (por como luego se desarrolla nuestro algoritmo

alcanzaría el mínimo de los 2).
Nuestra hipótesis es que alcanza con tomar ”CantIter” igual al doble del ancho de la matriz para tener buenas aproximaciones de los autovalores y autovectores.
Para confirmar esta hipótesis planteamos un experimento con una matriz generada de forma aleatoria de tamaño $100x100$. Calculamos sus autovalores con nuestro algoritmo (”autovalores calculados”) usando un ”CantIter” $= 200$ y los comparamos en un gráfico con los autovalores obtenidos por numpy (”autovalores reales”), graficando también la diferencia entre estos (escalada para poder visualizarse mejor).

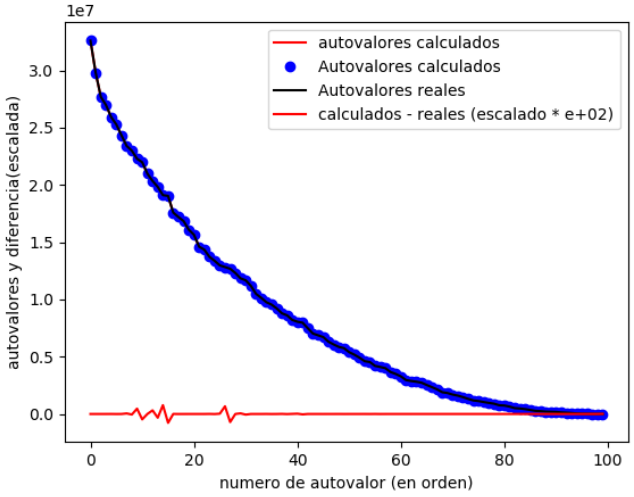


Figura 2: en negro autovalores calculados numpy. En azul autovalores calculados por nuestro algoritmo. En rojo la diferencia entre uno y otro, escalada por una constante para visualizarse mejor.

3.1.4. Observaciones:

Se observa que los autovalores calculados están suficientemente cerca de los autovalores reales (están sobre la misma linea y la diferencia es casi 0 en la mayoría de los casos).
Se observan también algunos picos en la linea de diferencia , estos picos coinciden con los momentos en que hay 2 autovalores de magnitud casi igual. Es decir que si la matriz tiene autovalores repetidos o muy similares (no nulos) nuestro algoritmo tendrá margenes de error mucho mas grandes. Este comportamiento es esperable ya el método de las potencias pide expresamente que el autovalor de mayor magnitud sea único, es decir que el segundo autovalor (en orden) no este cerca, cuanto mas cerca están el autovalor de mayor magnitud con el segundo mas tarda la convergencia, a efectos prácticos, mayor termina siendo el error.

3.1.5. Conclusiones del experimento 1:

El método de la potencia aproxima autovalores con una precisión que crece rápido en las primeras iteraciones. Cuanto mas grande sea la matriz, mas iteraciones se requieren pero incluso para matrices grandes, con una cantidad de iteraciones igual al doble del $max(ancho, largo)$ de la matriz es suficiente para una aproximación razonablemente buena.
Es inevitable que usando este algoritmo aparezcan errores cuando hay 2 autovalores prácticamente iguales o muy cercanos en magnitud. Por no ser un método particularmente rápido y no ser extremadamente preciso (incluso malo en el caso de autovalores repetidos) recomendamos buscar algún método alternativo para el calculo de autovectores y autovalores.

3.2. Experimento 2:

En este experimento buscamos medir la efectividad del método kNN para clasificación, en particular determinar de ser posible cual es el rango de k optimo, es decir cual es el que maximiza el accuracy.

3.2.1. Experimento 2-A:

Para responder esto, ejecutamos el algoritmo con diferentes sets de train y test (en particular usamos los sets del K-fold5 con todo el dataset de caras) variando el k y graficamos el promedio de accuracy (promediamos el accuracy de cada par de sets train/test) en función del k. Las barras de error muestran el desvío standard.

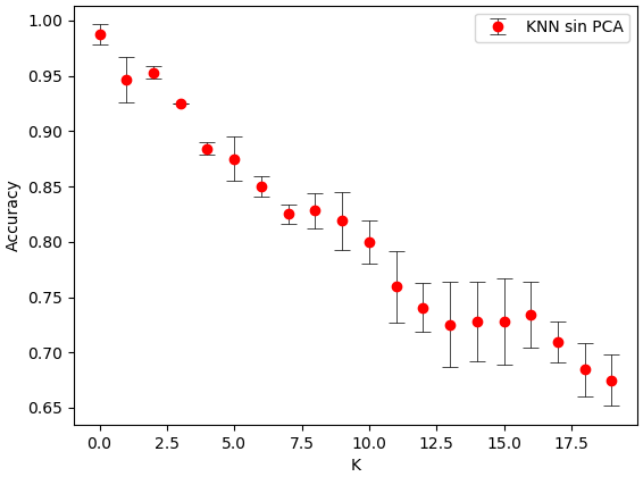


Figura 3: promedio del Accuracy del algoritmo kNNsin PCA en función del k elegido para diferentes particiones de todo el dataset de caras usando Kfold-5

3.2.2. Observaciones:

Se observa que el promedio de Accuracy del algoritmo para los diferentes sets de Kfold-5 disminuye a medida que el K crece.
Se observa también que este resultado dio similar para diferentes grupos de training-testing ya que la barras de error parecen mostrar que el comportamiento es el mismo en todos los grupos.
Por ultimo se discutió también que los resultados en un dataset considerablemente mas grande (muchas mas que 10 imágenes por sujeto) podrían ser sustancialmente diferentes. Un problema con este dataset tan chico es que hacer crecer el k en 1 unidad implica hacerlo crecer mucho, ya que 1 imagen es un 10 % del total de imágenes por persona (incluyendo las imágenes de test). Nuestra hipótesis para futura investigación es que el k optimo es un k chico dentro del 10 % de ancho del dataset.

3.2.3. Experimento 2-B:

Para esta segunda parte del experimento tuvimos la intención de desligarnos del método K-fold y evaluamos el comportamiento del algoritmo kNN sin PCA para diferentes K y aumentando el tamaño de set de Train.
En particular generamos conjuntos correspondientes de Train/Set en los cuales, el conjunto $Train_i/Test_i$ tiene en la parte de Train ($Train_i$) i imágenes y en la parte de Test ($Test_i$) $10 - i$ imágenes de cada persona. De este modo generamos 8 conjuntos con diferentes tamaños y graficamos sus Accuracys en función de k.

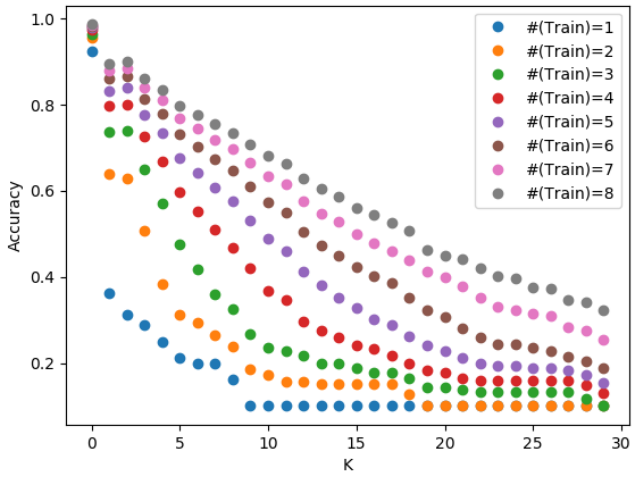


Figura 4: Accuracy del algoritmo kNN sin PCA en función del k elegido , para diferentes tamaños del set de Train

3.2.4. Observaciones:

Puede verse como el comportamiento es similar para todos los conjuntos de Train/Test, aunque en los casos que el set de Train es mas chico el Accuracy es menor en general.
Se observa entonces indirectamente el beneficio de tener un set de Train mas Grande.

3.2.5. Experimento 2-C:

Evaluamos el algoritmo del mismo modo que en el inciso A de este experimento pero esta vez evaluamos el comportamiento cuando crece el k a partir de la métrica f1-score, graficamos también como referencia, el Accuracy ya analizado en el inciso A.

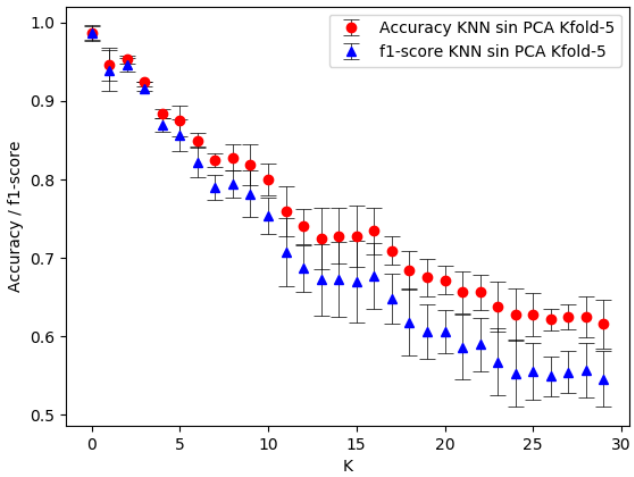


Figura 5: promedio del Accuracy del algoritmo kNNsin PCA en función del k elegido para diferentes particiones usando Kfold-5

3.2.6. Observaciones:

Observamos la correlación entre las 2 métricas, discutimos porque sucede así y estudiamos que es natural que suceda, en particular cuando el set de Test esta balanceado es decir, cuando hay una cantidad aproximadamente similar de imágenes a evaluar de cada categoría (cada cara), como es en este caso.
Como el dataset de caras no se presta a poder hacer experimentos desvalanceandolo mucho (por la limitada cantidad de caras por persona) decidimos dejar la experimentación de conjuntos desvalanceados para una futura investigación en la que se cuente con mas datos de cada categoría (cada cara).

3.2.7. Conclusiones experimento 2:

Concluimos a partir de lo observado que un k ideal será uno cerca de $k = 1$. Discutimos que para datasets más grandes el k óptimo crecerá en número pero se mantendrá proporcional en el porcentaje total que ocupa del dataset de cada sujeto. Es decir en este caso cada sujeto tiene 10 imágenes, $k = 1$ es un $10\,\%$ del conjunto. Suponemos probable que en un conjunto más grande el k crezca en magnitud pero se mantendrá siempre por debajo del $10\,\%$ del conjunto.
Se requiere repetir experimentación con un conjunto de datos más abundante para confirmar o rechazar esta hipótesis.
Por otro lado concluimos a partir del inciso B que el algoritmo funciona mejor cuanto mas grande sea el set de Train.

3.3. Experimento 3

En este Experimento nos proponemos hallar el $alpha$ optimo para la reducción de dimensiones.
Este alpha tendrá que ser lo suficientemente grande para devolver resultados comparables con el algoritmo kNN sin PCA y a la vez lo mas pequeño posible como para ahorrar computo.
Para establecer cual es el alpha que balancea mejor esto, ejecutamos el algoritmo variando el alpha dejando fijo la cantidad de iteraciones (en 300, aprox el doble del ancho, como fue sugerido en el experimento 1).El algoritmo lo ejecutamos con los diferentes conjuntos de Train/Test resultados de aplicar K-fold5 a todo el dataset.
Graficamos entonces los promedios de las diferentes métricas (Accuracy y f1-score) en función del alpha para estos conjuntos de Train/Test, con barras de error mostramos el desvío standard.

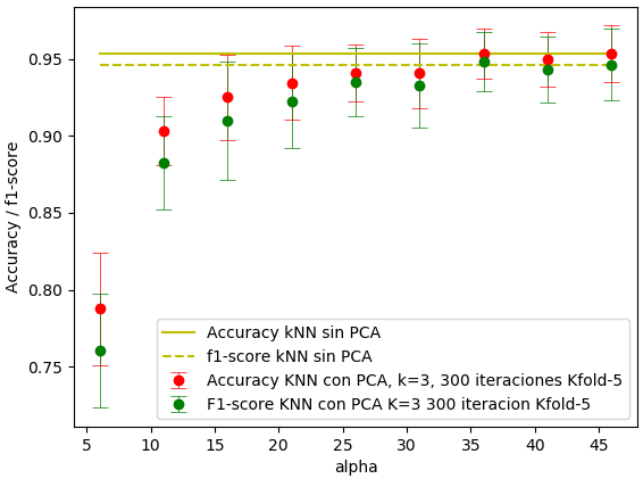


Figura 6: promedios de Accuracy y f1-score en función del alpha para los diferentes Sets de K-fold5

3.3.1. Observaciones:

Se observa del gráfico como ambas métricas correlacionan e incluso convergen al la eficiencia de kNN sin pca para aproximadamente el mismo alpha. se ve claramente como para alpha de tamaño mediano (mayor a 10) da resultados aceptables y como rápidamente a partir de $alpha = 25$ los resultados son similares a kNN sin pca y para $alpha = 35$ ya son indistinguibles, indicando así que el grueso de la información se concentra en relativamente pocas dimensiones.

3.3.2. Conclusiones Experimento 3:

Concluimos a partir de este experimento que un alpha apropiado para estas imágenes sera uno alrededor de 35 y si el tiempo de computo es critico podrían aceptarse alphas cercanos a 25 sin mucha perdida en la clasificación.

3.4. Experimento 4

En Este experimento nos proponemos medir los tiempo de computo del algoritmo.
Para la ejecución del Algoritmo kNN con PCA distinguimos 2 etapas, la etapa de reducción de las dimensiones de las imágenes en Train y la etapa de clasificar las imágenes de Test. Por esta distinción es que tenemos diferentes experimentos, unos en los que se evalúe solo el tiempo de clasificación y otro en el que se evalúe el tiempo total de ejecución (incluyendo el tiempo de reducción y clasificación).

3.4.1. Experimento 4-A

El primer análisis sera del tiempo tardado por imagen procesada, en este análisis no tendremos en cuenta el tiempo que tarda la reducción de dimensiones, solo el tiempo de clasificación (incluyendo la transformación de la imagen a analizar a la nueva dimensión, osea incluida la aplicación de Tc()). Utilizamos los diferentes conjuntos de Train/Test que se obtienen al aplicar k-fold5 sobre todo el dataset de caras. Utilizaremos como 300 iteraciones en el método de las potencias por lo ya explicado en experimentos anteriores y variaremos el alpha.

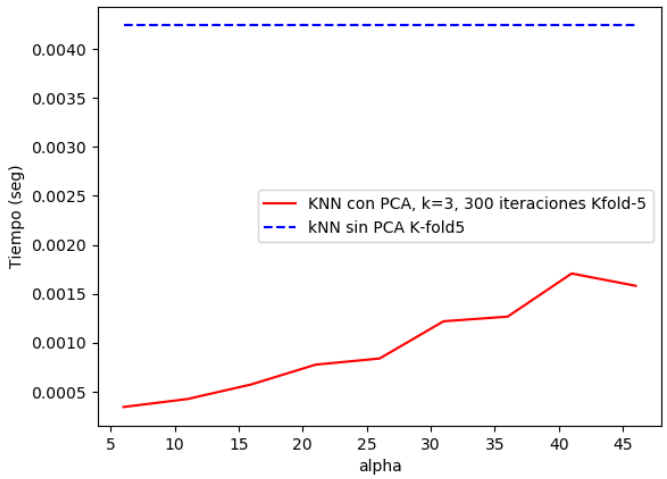


Figura 7: Promedio tiempos por imagen en función del alpha de kNN con y sin PCA, para los diferentes conjuntos de K-fold5.

3.4.2. Observaciones:

Observamos, como era de esperarse que el tiempo de procesamiento es considerablemente inferior para el algoritmo que ya aplico la reducción de dimensiones (PCA). Recordamos también del experimento 3 que con $alpha = 40$ no había diferencia aparente en ninguna de las 2 métricas con el algoritmo kNN

sin PCA, lo que sugiere mas conveniente utilizar el algoritmo con la reducción (con PCA) y $alpha = 40$ por sobre kNN sin PCA, pues este tardaría mas y el resultado seria igual(o muy similar).

3.4.3. Experimento 4-B

El análisis que corresponde ahora sera analizar la performance del los diferentes métodos teniendo en cuenta el tiempo que se tarda en la reducción.
Para esto repetimos el experimento del inciso A pero esta vez el tiempo lo calculamos desde el comienzo de la ejecución del código(incluimos la reducción de dimensiones).

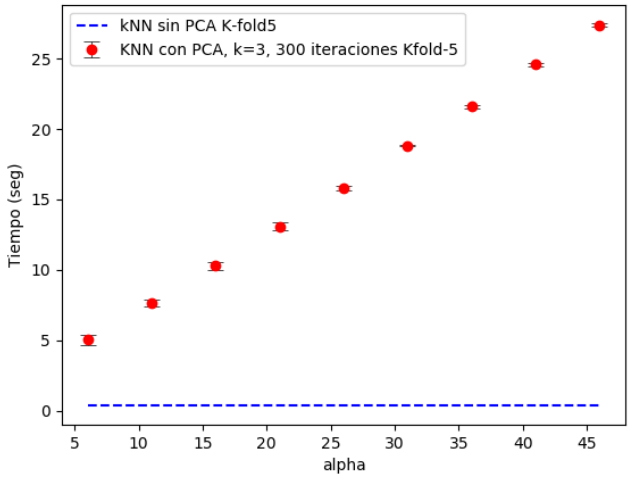


Figura 8: promedio Tiempos del algoritmo en función del alpha de kNN con y sin PCA, para los diferentes conjuntos de K-fold5.

3.4.4. Observaciones:

Puede observarse a simple vista lo mucho que tarda la reducción de dimensiones. Posiblemente por lo lento que es nuestro algoritmo para encontrar autovalores y autovectores.

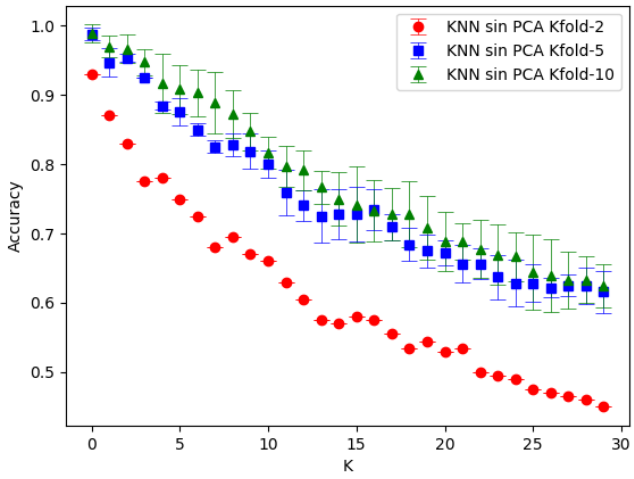
3.4.5. Conclusiones experimento 4:

Concluimos a partir de los observado que es recomendable utilizar el algoritmo kNN con PCA y alpha cerca de 40 en los casos que se cuente con tiempo (y poder de computo) suficiente para preprocesar el set de Train. De no contarse con la posibilidad de preprocesamiento del set Train (porque el set se actualiza en tiempo real y cambia mucho, por ejemplo) se sugiere utilizar el algoritmo Knn sin PCA que es un poco mas lento, pero tampoco es tan sustancial la diferencia como si lo seria reducir dimensiones en cada ejecución.

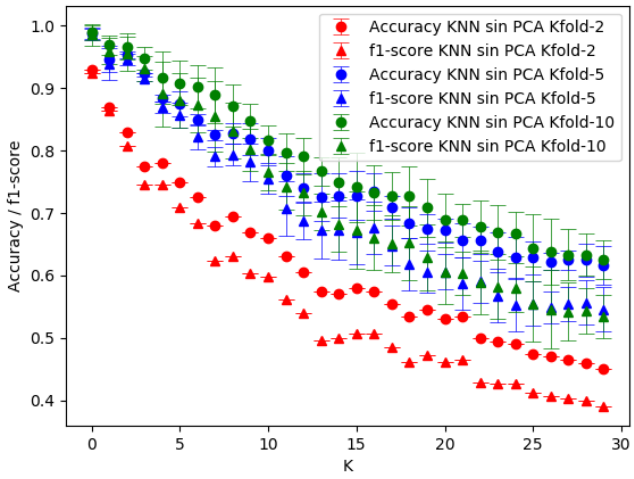
3.5. Experimento 5

Este ultimo experimento esta ligado a los experimentos anteriores y busca evaluar el metodo k-fold para generar instancias de Train y Test a partir de un dataset.
Para este experimento se ejecutaron los experimentos anteriores (al menos los que nos parecieron relevantes) con el fin de poder observar si el comportamiento es uniforme en las diferentes formas de particionar el dataset de caras y en caso de que no sea así evaluar porque.

3.5.1. Experimento 5: 2-A



3.5.2. Experimento 5: 2-C

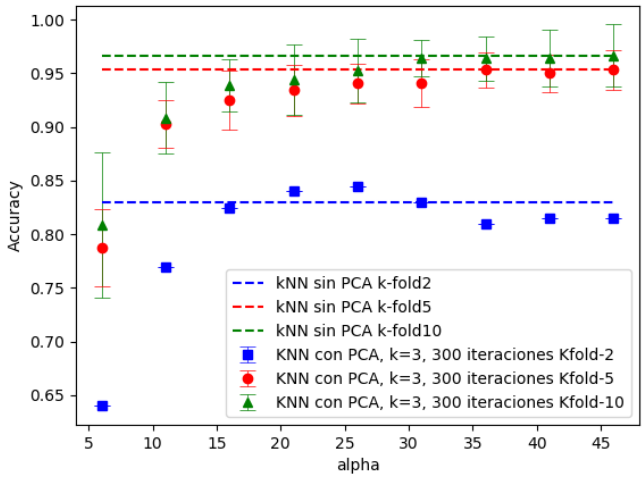


3.5.3. Observaciones:

Se observa como para ambas métricas el comportamiento es similar al mostrado en el experimento 2-A y 2-C.
Se observa también que los conjuntos generados por K-fold2 tiene barras de error muy chicas en comparación con los otros y su accuracy y f1-score es significativamente menor.
Por otro lado se ve que los de k-fold10 tiene un accuracy y f1-score ligeramente mayor que k-fold5 pero en contraposición sus barras de error son un poco mas grandes.

3.5.4. Experimento 5: 3

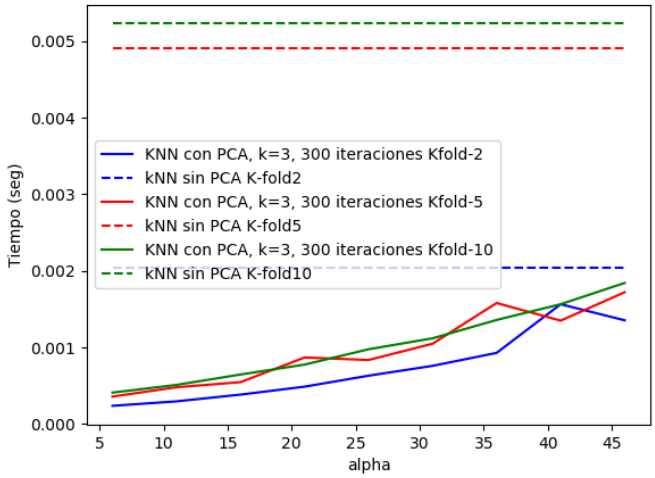
En este gráfico omitimos los f1-score ya que no permiten que se entienda bien el gráfico y a la vez no aportan mucha información, pues se correlacionan muy claramente con los accuracys.



3.5.5. Observaciones:

Se observa de nuevo comportamiento similar a lo planteado en el experimento.
Igual que en el inciso anterior se observa que los accuracys del algoritmo con los sets generados por K-fold2 son significativamente menor que los otros.
De nuevo se observa a k-fold10 con una ligera ventaja en accuracy por sobre el set generado con Kfold5 pero otra vez, es menos preciso en el sentido que sus barras de errores son mas grandes.

3.5.6. Experimento 5: 4-A



3.5.7. Observaciones:

Se ve como el comportamiento es similar al experimento 4 a excepción de el algoritmo kNN sin PCA cuando se usa k-fold2, este funciona mas rápido.

3.5.8. Conclusion experimento 5

Concluimos a partir de lo observado que el comportamiento es similar para las diferentes formas de particionar. No obstante la elección del k determinara el accuracy/f1-score obtenido y la precisión de los resultados obtenidos, sugerimos por lo tanto un k alto pero no al extremo de perder toda precisión (en

el sentido de las barras de error). Consideramos que el K-fold5 utilizado en muchos experimentos es una buena elección y es representativa de las demás.

4. Conclusiones

Concluimos de lo experimentado que una configuración optima para los parámetros dependerá de diferentes factores:
En todos los casos el **k asociado al algoritmo kNN** lo tomaríamos chico, en lo posible por debajo del 10 % del ancho del set de caras (es decir de la cantidad de caras por persona).Esto deriva directamente de las conclusiones del experimento 2.
-Si el caso de uso permite precomputar la reducción de dimensiones (PCA):
En este caso, en el que hay tiempo suficiente para computar la reducción de dimensiones previo a la ejecución del algoritmo, conviene utilizar este método (la reducción) pues incluso usando un alpha relativamente medio/alto ($alpha = 40$), el tiempo de clasificación por imagen es menor que el tiempo que requiere usar el algoritmo sin la reducción, mientras que para dicho alpha, tanto el accuracy como el f1-score son prácticamente iguales.
Posibles ejemplos de caso de uso como estos serán aquellos que el set de Train sea relativamente estático, por ejemplo reconocimiento facial para fichaje de empleados en una empresa.
El alpha que se elegirá dependerá del accuracy que se requiera necesario , en el ejemplo anterior (fichaje en una empresa) probablemente se priorice el accuracy al tiempo de ejecución por lo que se utilizara un $alpha >= 40$, mientras que quizás en otro caso de uso, como podría ser: identificar personas peligrosas en algún sitio de interés a partir de imágenes tomadas en tiempo real (buscar coincidencias en una gran base de datos preestablecida), convenga sacrificar Accuracy para ganar velocidad.En estos casos se podrá optar por un alpha cercano a 15 o incluso un poco menor manteniendo Accuracy y f1-score altos.
-Si el caso de uso NO permite precomputar la reducción de dimensiones (PCA):
En este caso sugerimos utilizar el algoritmo sin el método de reducción (sin PCA) pues las reducciones tardan mucho en realizarse y por otro lado, en las reducciones se pierde Accuracy y f1-score. Es decir, no tendria niguna ventaja y si desventajas. Posibles ejemplos de caso de uso serian aquellos en los que el set de Train esta en constante cambio , un ejemplo (ad hoc) podría ser un sistema de molinetes que deje pasar a una persona que acaba (1 segundo atrás) de registrar que pago la entrada en una caja. Lo esencial del ejemplo, es que la imagen de su cara (y el resto del las caras de las demás personas que quieren pasar) se acaba de registrar y casi inmediatamente es necesario utilizar esa información, y mas aun,invertir tiempo en procesar esa imagen no tiene sentido para un futuro porque la persona no va a volver a pasar por ahí.

Las conclusiones en ambos casos se extraen de las conclusiones de los experimentos 3 y 4.
El ultimo parámetro que queda por evaluar es el **"cantIter"** o **"cantidad de iteraciones del método de la potencia"**. Este parámetro sugerimos por lo concluido en el experimento 1 tomarlo cercano al doble del ancho (o largo,el máximo) de la imagen pues este valor da relativamente buenos resultados para computar autovalores y autovectores. Si se busca optimizar tiempo puede acortarse al simplemente tomarlo igual al ancho/largo de la matriz pero esto ya puede dar muchas mas excepciones (diferencia entre autovalor calculado y autovalor verdadero) de las deseadas. Aumentar este numero para ganar precisión no funciona, al menos no aumentarlo en una cantidad razonable pues ya para este valor: o bien los autovalores y autovectores calculados ya convergieron a los verdaderos o hay 2 autovalores muy cercanos en magnitud por lo que para que se de la convergencia se requerirá un incremento sustancial en "cantIter" (sustancial como "casi impracticable").
En cualquier caso, tanto para mejorar el tiempo como la calidad obtenida en la búsqueda de autovectores y autovalores, sugerimos a partir de lo experimentado explorar otros métodos de aproximación de autovalores y autovectores.

Del uso del metodo K-fold concluimos el K indicado es uno que genere sets De train lo suficientemente grandes pero que a la vez deje sets de Test no extremadamente pequeños para poder tener una evaluación mas informada. Si el set de train se reduce demasiado los resultados comienzan a ser no representativos

de un set de train real.
El método K-fold es particularmente útil para determinar la mejor opcion entre 2 o mas modelos (en este caso, kNN con y sin pca con diferentes alphas) ya que nos permite (como se realizo en los experimentos 2,3 y 4), ejecutar un algoritmo multiples veces con diferentes conjuntos de Train y Test y tomar promedios o utilizar otras herramientas estadisticas, todo a partir de un unico dataset (reducido) para luego comparar con otro algoritmo ejecutado y analizado estadisticamente del mismo modo.