Trabajo Práctico Especial 1

Reconocimiento de autocaras

Autores

Duffau, Teófilo Manuel (54151) tduffau@itba.edu.ar

D'Onofrio , Nicolas (54160) ndonofri@itba.edu.ar

Cifuentes, Ignacio (54311) icifuente@itba.edu.ar

Buscaglia, Matias (53551) mbuscagl@itba.edu.ar

Lynch, Ezequiel (54172) elynch@itba.edu.ar

93.75 - Métodos numéricos avanzados

Comisión S

14 de septiembre de 2018

Introducción

El avance tecnológico y el incremento exponencial año a año de la capacidad de procesamiento y almacenamiento de información han hecho que tecnologías como el reconocimiento facial en vivo sea algo utilizado por varias empresas para distintas funciones.

En este proyecto nos lanzamos a desarrollar varios métodos para un software que utilice el análisis de componentes principales con tecnologías de machine learning para ejemplificar un sistema de reconocimiento y clasificación de imágenes faciales.

Estructura de la implementación

Para la estructura nos basamos en los archivos ya provistos por la cátedra a los cuales añadimos un archivo *main* que se encargue de tomar la información sobre qué algoritmo utilizar y que tomará una imagen en vivo para decidir si esa persona fotografiada se encuentra en la base de datos.

El procedimiento de análisis toma las imágenes expresadas en formato pgm (escala de grises de 92x112 de tamaño) y las convierte cada una a un vector de 10.304 posiciones. Luego calcula la cara media y utiliza la estrategia pedida (sea PCA o KPCA) para encontrar las autocaras y así clasificar y medir precisión con las imágenes de testeo.

La diferencia entre PCA y KPCA es que la primera siempre encuentra componentes principales lineales para representar la data en menos dimensiones, cosa que no siempre es la mejor opción (la linealidad), por eso se utiliza KPCA, que logra conseguir componentes principales no lineales que según el problema pueden ser mejores que los lineales.

Para el cálculo de las matrices Q y R se implementó una función de control para hacer el corte de iteraciones llamada “isConvergingTriangular” que recibe una variable de error. Con la misma se busco dar control parametrizable sobre el margen de error. De esta manera cualquier número menor será considerado equivalente a cero lo cual facilitará cuentas y permitirá aproximar. Como no realizamos mayores optimizaciones sobre el cálculo de QR, si la matriz es grande, demora mucho tiempo en cortar a menos que el error sea relativamente grande, por lo que al final decidimos cortar el algoritmo de QR luego de x iteraciones. Al comparar los resultados con los valores obtenidos por las librerías oficiales de python no hubo mayores diferencias.

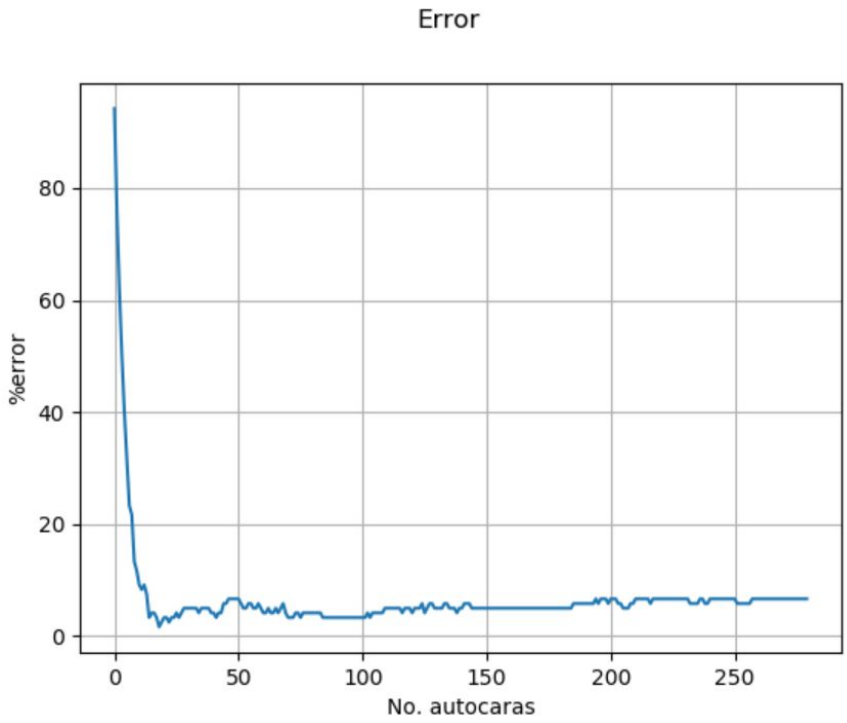
Descomposición en Valores Singulares

Debimos calcular una aproximación de los autovalores de AA^T y para esto utilizamos el algoritmo QR. Este algoritmo va iterando hasta triangularizar una matriz que al ser similar a AA^T tiene los mismos autovalores. Primero calculamos la factorización QR de AA^T y entramos a la iteración con una matriz igual a RQ. Luego en cada iteración se calcula la factorización QR de la RQ de la anterior utilizando Gram-Schmidt. Al ser matrices de gran tamaño iterar hasta que se haya triangulado resultaba en tiempos muy altos, por lo que decidimos cortar las iteraciones utilizando un máximo de 200. Como el algoritmo converge rápidamente al principio a valores cercanos a los reales pudimos ver testeando que esta cota no afecta significativamente el resultado obtenido, es decir, los autovalores.

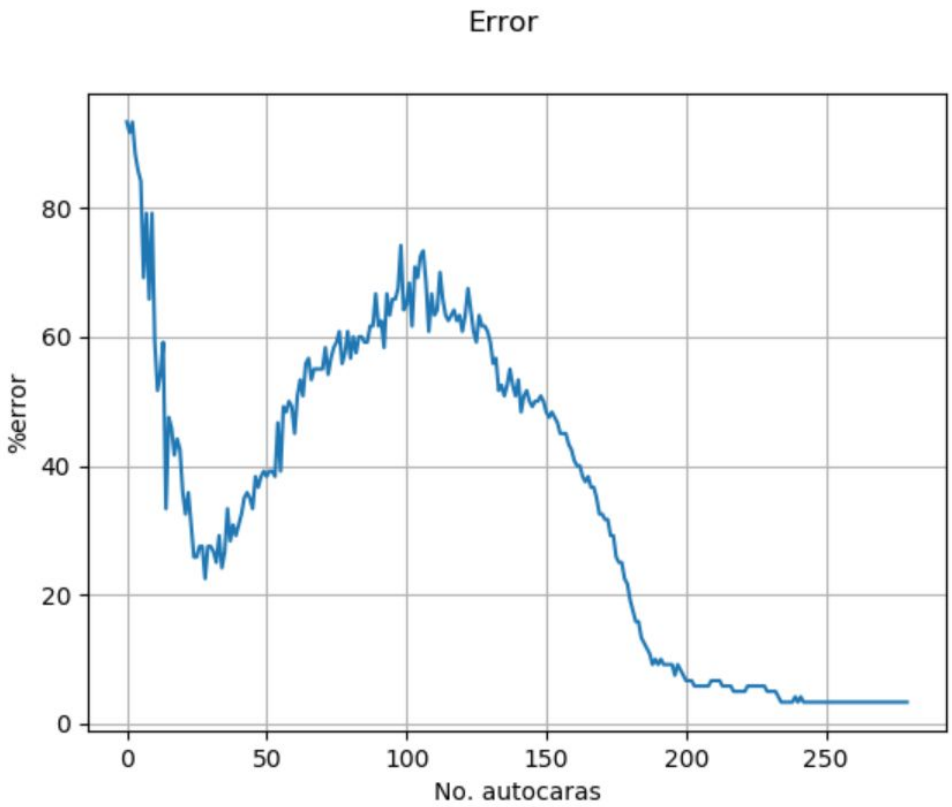
Análisis de los pruebas

Con el objetivo de probar las implementaciones como también comparar velocidades de cómputo y precisión, se llevaron a cabo seis pruebas con el código provisto por la cátedra pero invocando las implementación nuevas. Tres pruebas con la implementación PCA y otras tres para KPCA. El error se grafico utilizando la librería `matplotlib` y código de ploteo provisto por la cátedra permitiendo la comparación.

Para los dos sets de pruebas mencionados, la primera prueba de cada uno se realizó con 400 imágenes de cuarenta personas de una base de datos de AT&T. De las mismas, ocho fueron para entrenar y dos para realizar la prueba. Esta simulación se corrió una vez utilizando PCA y resultando en lo siguiente.



De la tabla anterior utilizando PCA se puede apreciar el avance del error con un tiempo de cómputo del algoritmo implementado de 73.80 segundos. (543 segundos contando el calculo de error)



De esta segunda tabla con KPCA se puede comparar el error con un tiempo de cómputo del algoritmo implementado de 78.73 segundos.(935 segundos contando el calculo de error)

Se puede apreciar la diferencia de tiempos entre un método y otro debido principalmente al Kernel trick que implementa el segundo método pero que por otro lado compensa por la velocidad en la que reduce el error. Para corroborar esto con un set de pruebas distinto se muestran pruebas similares con dos integrantes del equipo utilizando la webcam para ejecutar la búsqueda tras haberse sacado las imágenes para la base bajo las mismas condiciones.

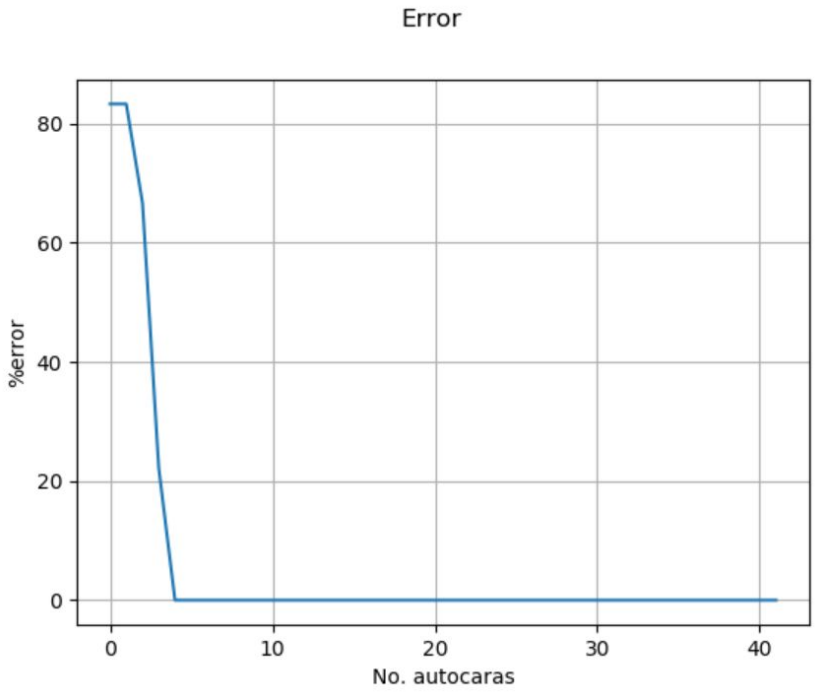


Gráfico de error del Integrante 1 utilizando PCA (tiempo de computo: 2778 ms)

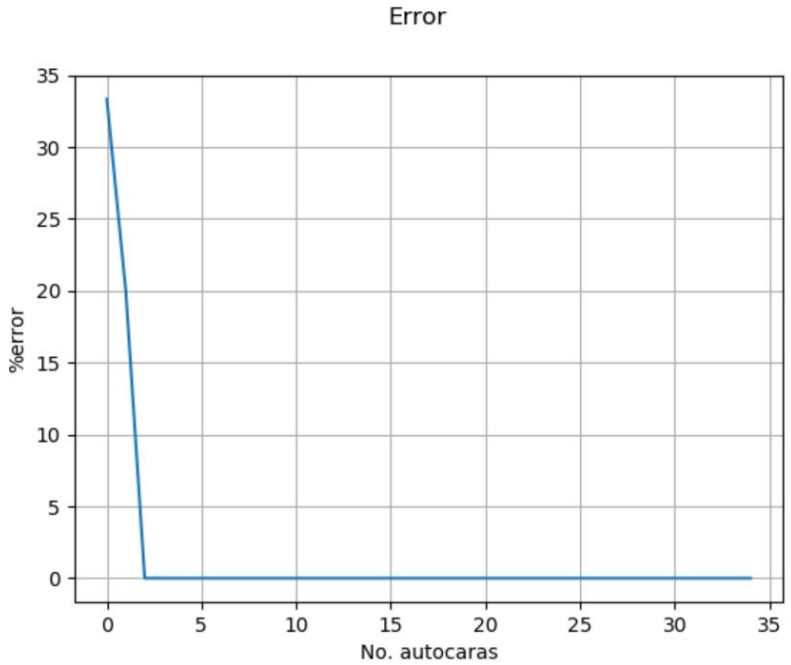


Gráfico de error del Integrante 2 utilizando PCA (tiempo de cómputo: 2551 ms)

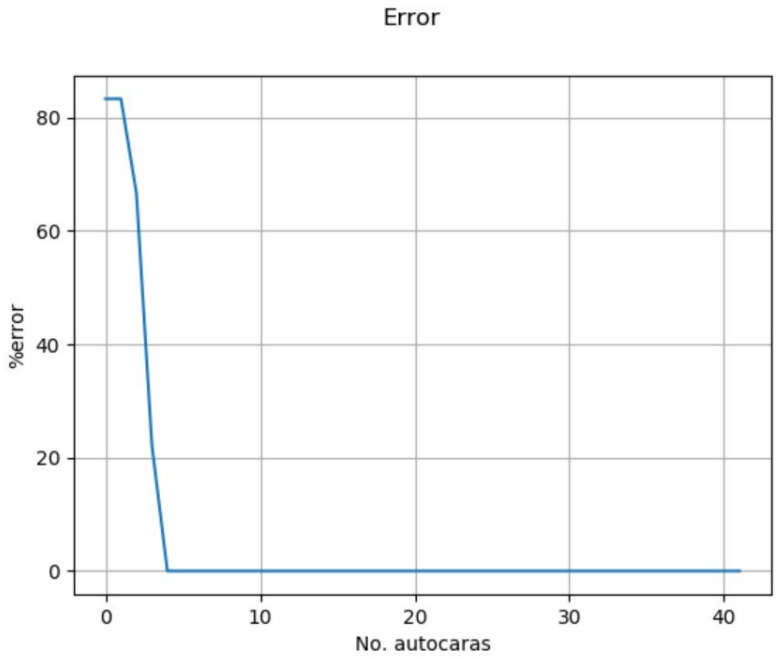


Gráfico de error del Integrante 1 utilizando KPCA (tiempo de cómputo: 1109 ms)

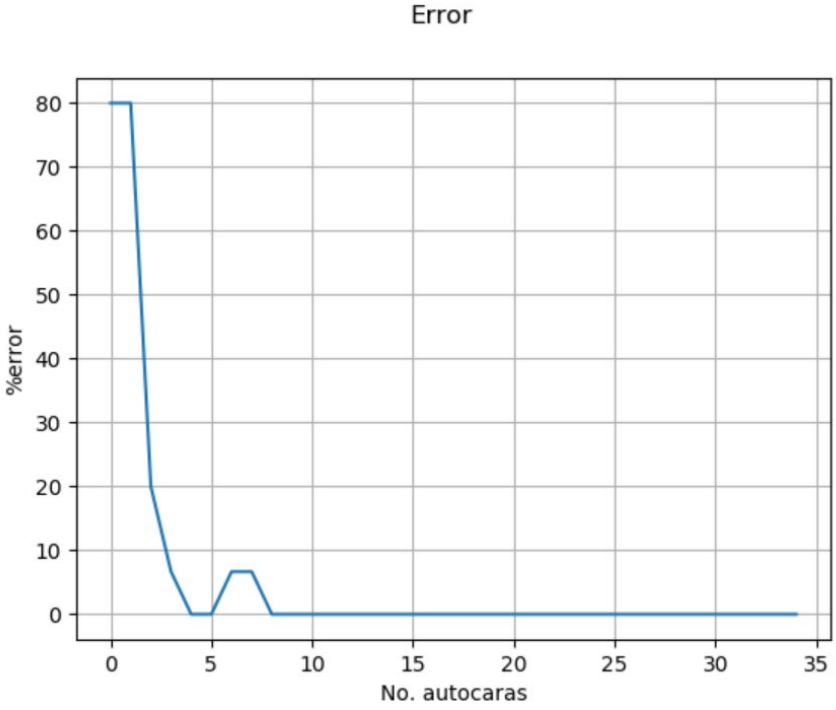


Gráfico de error del Integrante 2 utilizando KPCA (tiempo de cómputo: 1109 ms)

Conclusiones

Cabe destacar después de observar la información recopilada, que con las imágenes de prueba del grupo inicialmente hubo problemas probando los algoritmos. Tras algunas pruebas descubrimos una alta tasa de errores debido a que las imágenes habían sido sacadas en distintos lugares con distintos factores de luz y pose de los sujetos entre otros factores. Una vez descubierto esto y sacadas las fotos bajo las mismas condiciones optimas para todos se vio una considerable mejora.

Como mejora se podría realizar un preprocesamiento de las imágenes para evitar que el contexto afecte al reconocimiento, por otro lado, si se implementaran optimizaciones al cálculo de SVs, el algoritmo no demoraría tanto tiempo en procesar.

Referencias

1. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.383.6655&rep=rep1&type=pdf
2. https://eva.fing.edu.uy/file.php/514/ARCHIVO/2010/TrabajosFinales2010/informe_final_ottado.pdf
3. http://www.math.tamu.edu/~dallen/linear_algebra/chpt6.pdf
4. http://www.face-rec.org/algorithms/Kernel/kernelPCA_scholkopf.pd