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

Metodos Numericos 1C 2021

Trabajo Práctico 2: Reconocimiento de Numeros

Grupo 16

Integrante	LU	Correo electrónico
Cantini Budden, Sebastian	576/19	sebascantini@gmail.com
Seisdedos, Jose Luis	797/19	jlseisdedos11@gmail.com
Kruger, Lucas Ivan	799/19	lucaskruger10@gmail.com

Resumen:

En el presente informe buscamos estudiar una implementacion de OCR, comparando sus resultados en diversos escenarios y con el uso de un metodo que optimiza su performance. Implementamos el algoritmo de k-vecinos mas cercanos (kNN) como nuestro algoritmo de reconocimiento de números y utilizamos Principal Component Analysis (PCA) como el algoritmo que lo optimiza.

Palabras Clave

KNN, PCA, Aprendizaje Automático, Reconocimiento

I. INTRODUCCIÓN TEÓRICA:

En el presente informe vamos a estudiar algoritmos de Optical Character Recognition, los cuales buscan convertir imágenes con un número de un digito a su formato digital. El algoritmo a usar es el kNN (k Nearest Neighbors) y vamos a estudiarlo utilizando el conjunto de datos MNIST. Además vamos a comparar los resultados con otra implementación que agrega PCA y tomando diferentes cortes de los sets de datos usando la herramienta kFold.

A. *kNN:*

El algoritmo de **kNN** recibe como input un conjunto de datos con un valor al que representan, y además otro conjunto de datos a los que tiene que clasificar o dar un valor en base a los del primer conjunto. Para kNN interpreta a cada dato de entrada como un punto $\in R^m$ donde m son la cantidad de variables a tomar en cuenta de cada dato de entrada, en este caso la cantidad de píxeles de cada imagen. Una vez que tenemos todos los puntos y el valor que representan, podemos tomar el nuevo dato a clasificar, lo convertimos en un punto y nos quedamos con los k vectores a distancia más cercana. De estos k nos quedamos con el valor que tenga mas apariciones y devolvemos este como nuestra reconocimiento. Para encontrar los puntos mas cercanos utilizamos la distancia euclidiana, es decir restamos los puntos y les tomamos la norma 2. Dependiendo del valor de k el valor obtenido puede variar, por eso es importante estudiar esta variable.

B. *PCA*

Los sets de datos con gran tamaño son difíciles de interpretar y computacionalmente costosos. Para esto es conveniente utilizar la técnica **PCA** (Principal Component Analysis) la cual busca reducir la dimensionalidad de dichos sets de datos, incrementando la interpretabilidad pero a la vez minimizando la perdida de información.

Para esto debemos realizar los siguientes pasos:

- **Restar promedio de columnas:** Sea D la matriz de R^{mxn} donde m son la cantidad de coordenadas de cada elemento en el espacio y n la cantidad de elementos (en nuestro caso m = píxeles y n = n-esima imagen). Sea $0 \leq i < m$ y $0 \leq j < n$. Sea U una matriz de R^{mxn} donde

$$U_{ij} = \sum_{k=0}^{n-1} \frac{D_{kj}}{n} \tag{1}$$

  Es decir, para cada elemento de una columna este tiene el valor del promedio de la misma. Definimos X como:

$$X = D - U \tag{2}$$

  Siendo X la matriz con la resta del promedio de columnas.
- **Obtenemos la matriz de covarianza:** Para esto basta con realizar la siguiente operación:

$$C = \frac{1}{n-1} X^t X \tag{3}$$

  Esta matriz tiene las varianzas en la diagonal y las covarianzas en las demás posiciones.
- **Diagonalización de matriz** La matriz C es cuadrada y simétrica. Entonces podemos asegurar que podemos escribirla como:

$$C = PC'P^t \tag{4}$$

  Donde C' es la matriz diagonal con los autovalores ordenados de mayor a menor en modulo (de izquierda a derecha). Y P es una matriz que tiene los autovectores asociados a los autovalores (y con norma 1) como columna, ordenados según su autovalor de izquierda a derecha. Entonces como las columnas son ortonormales y de norma 1 sabemos que P es ortogonal y entonces su inversa es P^t. Es importante notar que al diagonalizar D estamos buscando variables que tengan covarianza 0 entre sí y la mayor varianza posible.
- **Cambiamos de base:** Ahora solo resta realizar el siguiente cambio de base

$$D' = DP \tag{5}$$

  Donde D' es nuestra matriz final.

Nuestra matriz resultante D' va a estar ordenada por las columnas que maximizan la varianza del conjunto de datos. Entonces, no es necesario conservar todas las columnas, permitiéndonos reducir su dimensionalidad, solo preservando las primeras α columnas. Este factor α sera estudiado más adelante en la experimentación.

C. *Método de Potencia*

El método de potencia, es un algoritmo que sigue una secuencia con el fin de aproximar el autovector de una matriz asociado con el autovalor mayor de la misma matriz.

Dado un $q^0 \in \mathbb{R}^n$ con $||q^0||_2 = 1$, entonces la sucesión q^k (mencionada anteriormente), esta definida de la siguiente manera, Para k = 1,... :

$$z^k = Aq^k - 1 \tag{6}$$

$$q^k = \frac{z^k}{||z^k||_2} \tag{7}$$

Donde A es nuestra matriz. Esta secuencia logra que q^k converja al autovector[1] y además, $\lambda_k = (q^k)^t A q^k$ converge a λ_1. Con $\lambda_1 > \lambda_2, ..., \geq \lambda_n$. Así podemos utilizar este método para conseguir el autovalor mayor y su autovector asociado.

D. *Método de Deflación*

Dado una matriz simétrica A, con A $\in \mathbb{R}^{nxn}$, y con autovalores $\lambda_1 > \lambda_2 > ... > \lambda_n$ entonces el método de deflación nos indica que si le aplicamos la próxima ecuación:

$$B = A - \lambda_1 * v_1 * v_1^t \tag{8}$$

Entonces podemos decir que los autos valores de B son $0, \lambda_2, ..., \lambda_n$. Notar que λ_1 y v_1 son el autovalor mayor y el vector asociado respectivamente.

E. *Métricas utilizadas*

- **Kappa de Cohen:** Es una métrica utilizada en la estadística que mide la concordancia entre dos resultados. Resulta útil obtenerla ya que lo realiza teniendo en cuenta el acuerdo que ocurre por azar. La ecuación para k es:

$$k = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \tag{9}$$

  donde Pr(a) es el acuerdo entre resultados y Pr(e) el generado hipotéticamente por azar.
- **F1:** Mide la precisión que tiene un test. Se considera como una media armónica que combina valores de precisión y recall. La ecuación consiste en:

$$F1 = 2 * \frac{precisión * recall}{precisión + recall} \tag{10}$$

  Precisión mide la calidad de un modelo en tareas de clasificación, y sirve para reducir la cantidad de falsos positivos.
- **Accuracy:** Mide el porcentaje de casos que el modelo acertó. Es una métrica engañosa, ya que puede hacer creer que un modelo malo funcione mejor de lo que debería, en especial en casos donde las clases estén desbalanceadas. Su ecuación es:

$$\frac{Aciertos}{Casos} \tag{11}$$

- **Recall:** Mide el porcentaje de casos que el modelo es capaz de identificar. Lo hace mediante esta ecuación:

$$Recall = \frac{\tilde{C}+}{C+} \tag{12}$$

  donde $\tilde{C}$+ son los casos clasificados como positivo y C+ son los verdaderamente positivos. Es útil para reducir la cantidad de falsos negativos.

F. *K-Fold*

K-Fold es una herramienta que nos permite utilizar el método cross-validation para estimar el nivel de un algoritmo de aprendizaje. Esta herramienta fragmenta nuestro set de datos en k particiones. Luego, se toma una partición como test y se corre usando todas las otras como train. Esto se repite para cada partición y finalmente, para conseguir las métricas se toma el promedio de las k evaluaciones.

II. DESARROLLO

Implementamos tres algoritmos en C++: kNN, PCA y un algoritmo que busca autovalores y autovectores. El programa principal debe tomar al menos cuatro parámetros por línea de comando con la siguiente convención:
$./tp2 - m < method > -i < train\_set > -q < test\_set > -o < classif >$

Donde:
-$< method >$ es el método a ejecutar (0: kNN, 1:PCA+kNN).
-$< train\_set >$ es el nombre del archivo entrada con los datos de entrenamiento.
-$< test\_set >$ es el nombre del archivo con los datos de test a clasificar.
-$< classif >$ es el nombre del archivo de salida con la clasificación de los datos de test de $< test\_set >$.

La explicación de los tres algoritmos nombrados anteriormente será desarrollada abajo:

A. *kNN*

Para implementar este algoritmo utilizamos el esqueleto proporcionado por la cátedra, es decir, la clase "KNNClassifier".
Esta contiene en su apartado público tres funciones:

- KNNClassifier: la cual inicializa la clase y guarda el k. fit: recibe la matriz train sin la primer columna de valores, y un vector (la columna de train con los valores que separamos). predict: es importante notar que tanto la matriz de test como de train tienen una imagen en cada fila y que el vector de valores tiene en los mismos índices el valor correspondiente.
  Por cada imagen en test:

  - copiamos la matriz de train en D, y le restamos a cada una de sus filas la fila de la imagen actual de test. Esto nos deja en cada fila la resta entre ambos.
  - tomamos la norma euclidiana de cada fila de D y la guardamos en un vector (podria haberse usado la norma al cuadrado), de forma tal que los índices de este se relacionen con los de la matriz D.
  - creamos un vector con índices.
  - ahora ordenamos el vector de índices en función de las distancias calculadas anteriormente de menor a mayor.
  - utilizamos el vector de índices reordenado para hallar los k primeros valores correspondientes.
  - ahora nos quedamos con el valor que tiene mas apariciones entre éstos.
  - guardamos este valor en un vector resultado y volvemos a iterar..

En su apartado privado tres variables:

- kVecinos: un entero que guarda cuantos vecinos vamos a tomar en cuenta.
- D: la matriz de test sin la primer columna values: el vector que es la primer columna de la matriz test con los valores.

B. *PCA*

Para implementar este algoritmo utilizamos el esqueleto proporcionado por la cátedra, es decir, la clase "PCA".

Esta contiene en su apartado público tres funciones:

- PCA: la cual inicializa la clase y guarda el alpha.
- Fit: recibe la matriz train sin la primer columna (valores) y aplica los siguientes pasos:
  - Creamos un vector de largo igual a la cantidad de columnas de train, donde guardamos el promedio de sus columnas en cada posición.
  - A cada fila de la matriz train le restamos el vector creado y dividimos en cada posición por $\sqrt{n-1}$ siendo n la cantidad de filas de la matriz.
  - Llamando a la matriz resultante 'D', creamos $X = D^t D$
  - Usamos la función "get_first_eigenvalues" pasando como primer parámetro a X, como segundo a alpha, el tercero 5000 y el cuarto $1e^{-16}$. Guardamos el valor en la variable privada "eigenVectors".
- Transform: recibe la matriz de train y devuelve XE siendo E la matriz "eigenVectors"

En su apartado privado dos variables:

- alpha: guarda el valor de alpha recibido al inicializar.
- eigenVectors: es una matriz que tiene una cantidad α de autovectores como columnas.

C. *Método de Potencia/Deflación*

Se utilizo el método de potencia en combinación con el método de la deflación para implementar nuestro algoritmo que busca los autovalores y autovectores. power_iteration es nuestra función que devuelve un único autovalor y autovector asociado, este utiliza el método de la potencia.

Lo que hace esta función es generar un vector al azar al que normalizamos. Esto es posible ya que no importa con que vector comencemos. Sabemos que la serie $\frac{AV}{\|V\|^2}$ tiende al autovector asociado al mayor autovalor en modulo. En cada iteración realizamos la multiplicación y normalización, nos guardamos el resultado anterior para luego comparar el ángulo (entre el nuevo vector y el vector que se obtuvo antes), buscando que la diferencia sea menor a épsilon, un numero muy chico ($1e^{-16}$), con el fin de ver que ya se aproximo lo suficiente. Esto lo hacemos repetida mente creando una sucesión de vectores que, por el método de potencia, aproxima al autovector asociado con el autovalor mas grande y su autovector asociado.

La otra función, get_first_eigenvalues devuelve los k (parámetro de entrada num) autovalores mayores con su autovector asociado. Esta utiliza la función explicada anteriormente, para conseguir el autovalor mayor y su autovector asociado, luego le aplicamos la ecuación (8) a la matriz con el fin de poder volver a usar power_iteration y poder conseguir el próximo autovalor máximo y su autovalor asociado.

III. Resultados y Discusión:

Hipótesis:

1) Para mayor k, mejores predicciones: Al aumentar nuestro k esperamos obtener mejores predicciones ya que tomamos en cuenta a mas vecinos.
2) Para mas cortes, mejores predicciones: Debido al funcionamiento de k-Fold[referencia a la pagina de k-Fold], al realizar mas cortes vamos a tomar una parte mas pequeña de nuestro set de test y una parte mas grande para el de train, esto pensamos que llevaría a una mejor predicción.
3) Para un α mayor, mayor tiempo y mejor predicción: debido a que PCA se queda con α columnas, es decir en cierta forma comprime la

información de nuestra matriz de train, esperamos que al aumentarlo tengamos un mayor tiempo y una mejor predicción (ya que la matriz resultante es mas grande y que tiene mas información).

4) kNN vs kNN+PCA: esperamos una mejora en el tiempo al utilizar kNN+PCA debido a la reducción de columnas y que los cambios en las métricas no sean significativos, debido a que PCA se queda con la información relevante.

Experimentos:

- Estudio de k: usando un set de datos reducido (3000) probamos kNN con k de 2 a 100 (Este experimento busca estudiar la primer hipótesis).
- Estudio de Cortes: usando el set reducido y con kNN probamos con distinta cantidad de cortes, de 2 a 40, con la herramienta k-Fold (Este experimento busca estudiar la segunda hipótesis).
- Estudio de alpha: con el set reducido y con kNN+PCA probamos distintos alphas de 1 a 40 (Este experimento busca estudiar la tercer hipótesis).
- Estudio diferentes tamaños kNN: probamos los tamaños 6000,12000,18000,24000,30000,36000,42000 de set de datos con kNN.
- Estudio diferentes tamaños kNN+PCA: probamos los tamaños 6000,12000,18000,24000,30000,36000,42000 de set de datos con kNN+PCA.
- kNN vs kNN+PCA: comparamos los resultados (este experimento en conjunto a los dos anteriores buscan estudiar la ultima hipótesis).

A. *Estudio de k:*

Para esta experimentación usamos un conjunto reducido de los datos, debido a limitaciones técnicas, se tomaron 100 muestras de diferentes k. Con estas corrimos el algoritmo de kNN, usando 10 cortes. Con los resultados obtuvimos las métricas explicadas en la introducción teórica, y además el tiempo total de ejecución incluyendo los cortes.

Como podemos observar las métricas de accuracy, Kappa de Cohen, F1 y Recall toman gráficos muy similares, por lo que nos enfocaremos en fig.1. En este gráfico se observa primero un descenso de la curva, luego un pico cuando k es igual a diez (linea roja) y luego un descenso constante a medida que k aumenta.

Los datos contradicen nuestra primer hipótesis, es importante notar que en un principio si se cumple ya que se vuelve mas estable al agarrar más vecinos, pero luego deja de serlo. Esto se debe a que al aumentar la cantidad de vecinos mas cercanos, aumentamos la distancia para la cual se seleccionan vecinos, disminuyendo la precisión y aumentando la probabilidad de obtener anomalía en el grupo de vecinos cercanos. Esto es mas notorio en casos particulares como los manuscritos del ocho y el cero, ya que ambos números comparten rasgos particulares.

Con respecto al tiempo (fig.9.) podemos ver que hay una tendencia a bajar, sin embargo, este experimento fue realizado en otras ocasiones obteniendo gráficos muy dispares. También es importante notar, que el rango donde los valores se mueven no es muy grande. Esto puede deberse a que el algoritmo tiene un costo computacional similar en todos los casos y a la vez se ve afectado por otros procesos ejecutados en segundo plano. Observamos que el mejor k que podemos elegir es k=10 debido a que en todos los gráficos vemos un pico de mejora, pero estaría bien con un rango entre ocho y doce.

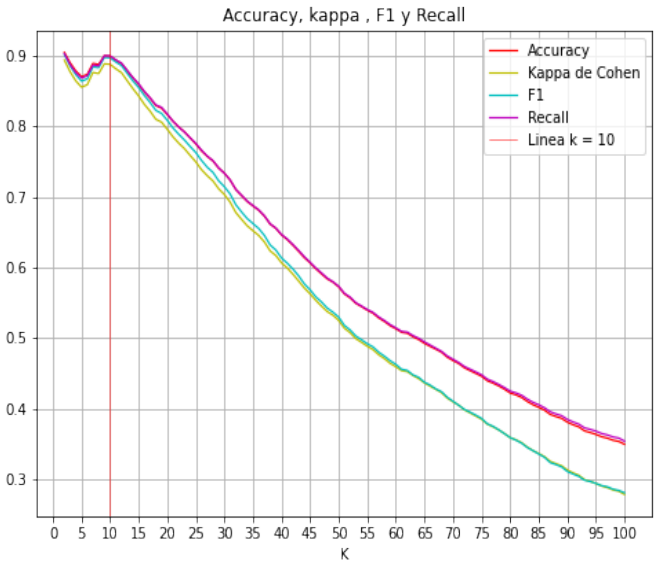

Fig. 1. Experimento donde se miden las métricas de distintos k.

B. *Estudio de Cortes:*

Para esta experimentación usamos un conjunto reducido de los datos, debido a limitaciones técnicas, se tomaron 40 muestras con diferentes C, siendo C la cantidad de cortes a realizar con el método k-Fold. Con estas corrimos el algoritmo de kNN, usando diez vecinos mas cercanos (este numero no es al azar, y es explicado en la sección de arriba). Con los resultados obtuvimos las

métricas explicadas en la introducción teórica, y además el tiempo total de ejecución incluyendo los cortes.

En este caso podemos observar en fig.2. valores similares en accuracy, Kappa de Cohen, F1 y Recall. Observamos que en casi todos los casos, los valores se estabilizan en 0.9 (variando alrededor del 0.01) a partir del valor C=15. Respecto al análisis del tiempo podemos ver que este aumenta de forma logarítmica. Esto puede deberse a que el aumento de los sets de entrenamiento es logarítmico. (fig.10.) Los datos observados otra vez

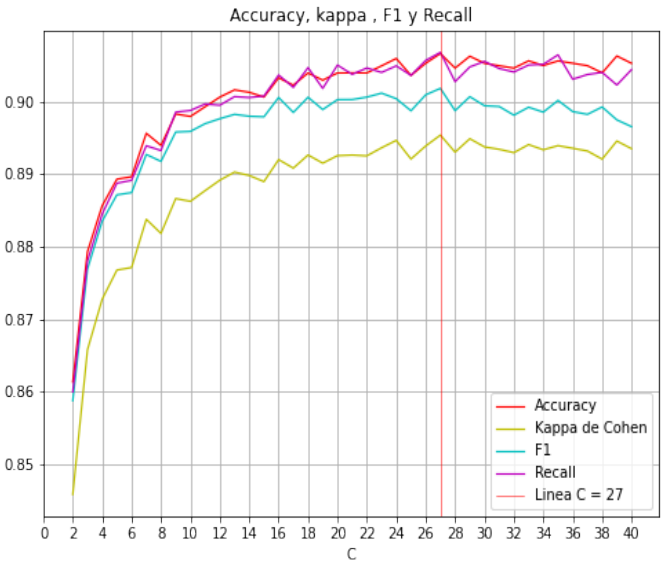

Fig. 2. Experimento donde se miden las métricas de distintos cortes.

contradicen parcialmente nuestra hipótesis. Vemos que entre k igual a dos y k igual a cinco, el aumento de cortes lleva a una mejora aunque teniendo en cuenta la escala, esta es despreciable. Luego se estabiliza. Esto se debe a que las clases están bien balanceadas en nuestro set de datos, provocando que la mejora por k-Fold no sea tan notoria. Esto lo podemos ver en nuestra notebook[2]. Es importante notar, que en casos donde los datos se encuentren desbalanceados, es mas probable que la hipótesis propuesta pueda cumplirse, debido al funcionamiento de k-Fold. Proponemos como mejor C a C igual a veintisiete debido a que en todos los gráficos hay un máximo local bastante marcado en ese punto. Aun así, es importante aclarar que podríamos tomar casi cualquier C a partir del dieciséis, ya que al observar las escalas vemos que las diferencias son muy chicas. El criterio usado para elegir 27 es que parece ser el máximo y el tiempo no tiene un gran aumento, alrededor de medio segundo.

Consideramos mas conveniente utilizar k-Fold en aquellos casos donde nuestro set de datos esta muy desbalanceado, ya que este método realiza el test con diferentes cortes y luego los promedia, obteniendo medidas mas confiables.

C. *Estudio de alpha:*

Para esta experimentación usamos un conjunto reducido de los datos, debido a limitaciones técnicas, se tomaron 40 muestras con diferentes alpha, siendo alpha la cantidad de componentes principales a tomar. Con esto corrimos el algoritmo de kNN+PCA, usando diez vecinos mas cercanos. Con los resultados obtenidos en fig.3. las métricas explicadas en la introducción teórica, y además el tiempo total de ejecución. Observando los resultados obtenidos, se prueba que nuestra hipótesis sobre alpha es verdadera. Sin embargo, se nota que a partir del alpha igual a catorce, no se observan mejoras significativas en las métricas. Esto indica que a partir de dicho alpha, la información que sigue deja de ser relevante, lo cual indica que es el alpha óptimo para PCA. Además a medida que alpha aumenta, el tiempo de cómputo también lo hace(fig.11.) . Esto es completamente esperable ya que se debe realizar el cálculo de más autovalores, por lo tanto más corridas de nuestra función que lo opera. La tendencia observada es que para mayor alpha, más tiempo en calcular el próximo, tal como proponemos en nuestra hipótesis.

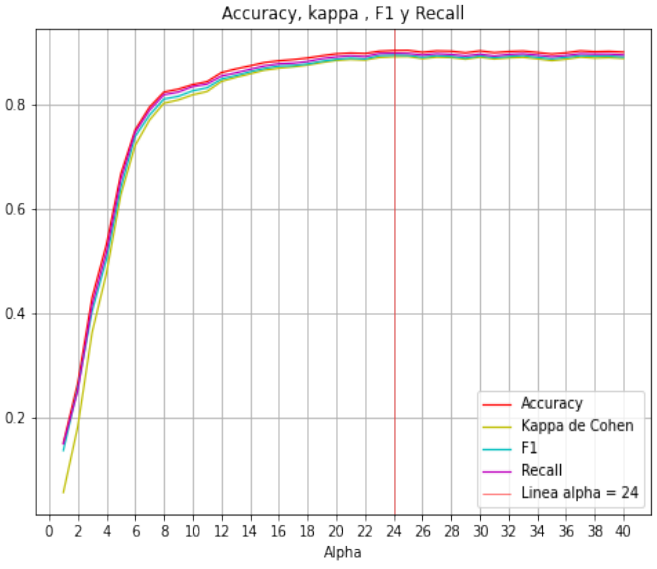

Fig. 3. Experimento donde se miden las métricas de distintos alpha.

D. Estudio de tamaño:

En esta experimentación corrimos el algoritmo de kNN y kNN+PCA con k-Fold en los tamaños de 6000, 12000, 18000, 24000, 30000, 36000, 42000 imágenes como base de datos de testeo, de las cuales solo podemos asegurar que el caso 42000 está balanceado(por el funcionamiento del muestreo). Tomando los 10 vecinos mas cercanos y 27 cortes. Con los resultados obtuvimos las mismas métricas mencionadas en los experimentos anteriores.

1) kNN:: Podemos observar, como en fig.4. las varianzas de los promedios obtenidos, tienden a disminuir con el aumento de la cantidad de imágenes(esto puede deberse a que tenemos mas muestras). Además pareciera que los valores aumentan mas rápido al principio (esto se debe a que). El tiempo parece aumentar de forma lineal, lo cual tiene sentido ya que el tamaño de nuestro set aumenta linealmente y nuestro algoritmo tiene complejidad lineal.

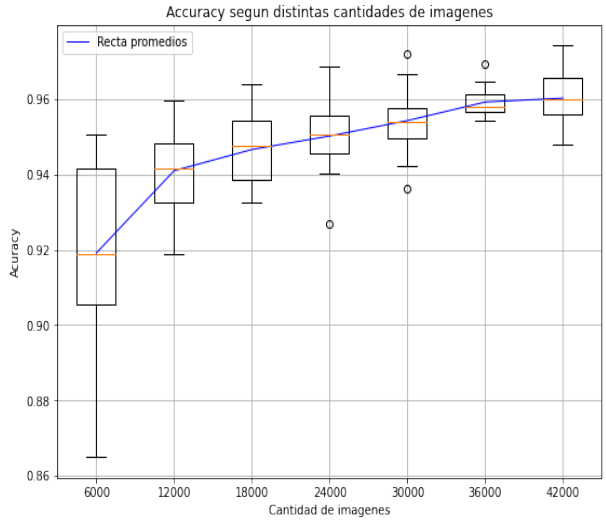

Fig. 4. Experimento donde se midió el accuracy de distintos tam. usando el algoritmo de kNN.

2) kNN+PCA:: Al igual que en kNN, se tiene que a mayor tamaño de imagen, menor dispersión entre las métricas (fig.12.) . Las observaciones para los tamaños de 6000 a 42000 son muy similares que kNN. Cuando comparemos ambos algoritmos veremos qué conclusiones obtener a partir de esto.

3) kNN vs kNN+PCA: Respecto a las métricas, podemos ver en ambos, fig.5. y fig.6., un aumento en ambos casos a medida que aumenta el tamaño. En algunos casos los datos usando PCA son mayores, como se puede ver en el accuracy en fig.7, y en otros usando solo kNN.

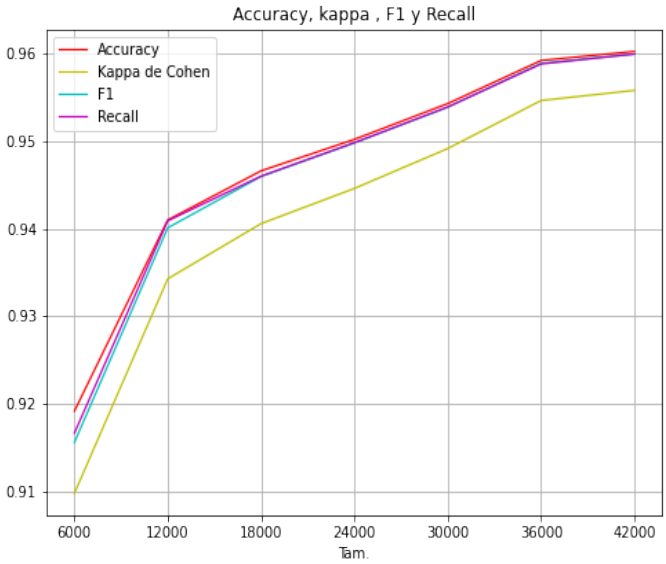

Fig. 5. Experimento donde se miden las métricas de conjuntos de train con distinto tamaño del algoritmo de kNN.

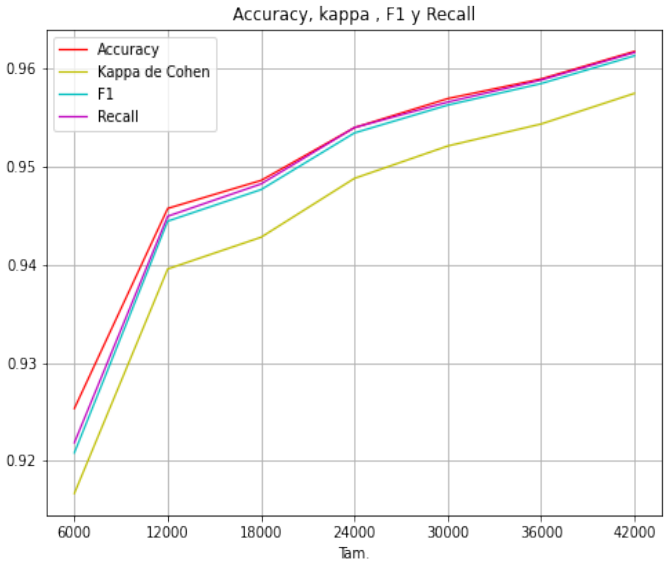

Fig. 6. Experimento donde se miden las métricas de conjuntos de train con distinto tamaño del algoritmo kNN en conjunto de PCA.

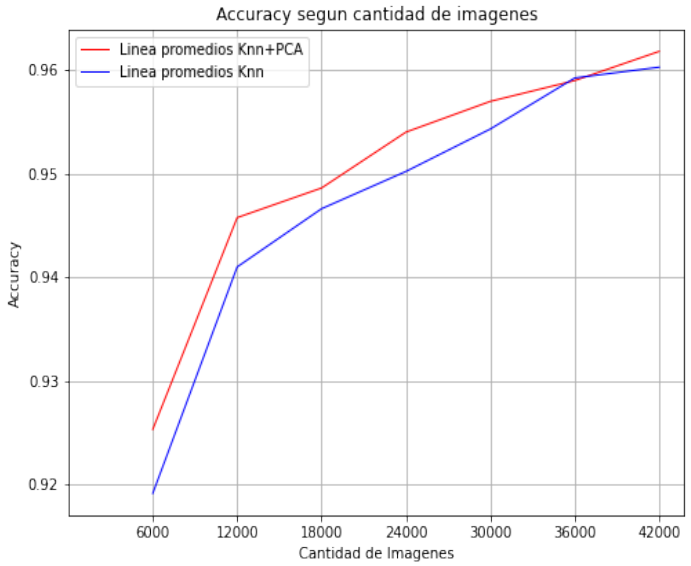

Fig. 7. Experimento donde se miden las métricas de conjuntos de train con distinto tamaño del algoritmo kNN en conjunto de PCA.

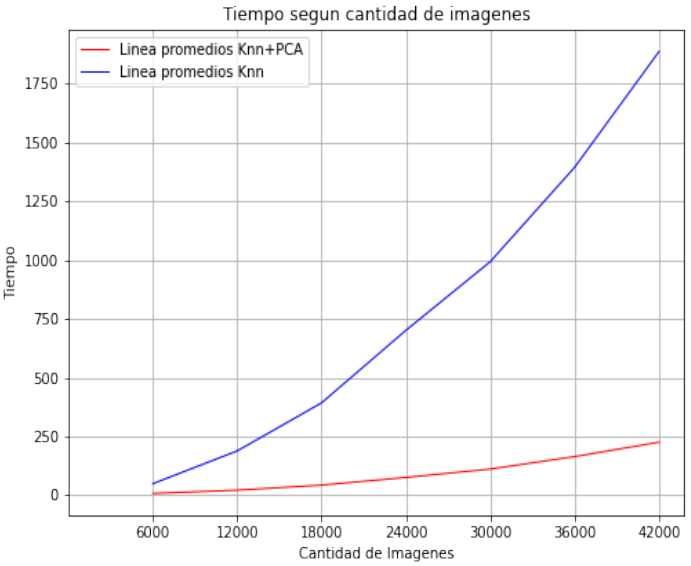

Fig. 8. Experimento donde se miden las métricas de conjuntos de train con distinto tamaño del algoritmo kNN en conjunto de PCA.

Se puede observar en fig.8., una ventaja considerable del tiempo del algoritmo en conjunto con PCA sobre el de kNN exclusivo, esto se debe a que la reducción que ofrece PCA permite un procesamiento mas rápido de los parámetros de entrada hecho por kNN. Respecto a las métricas, también gana PCA, consideramos que esto puede deberse a que la información omitida por PCA causa errores de medición en kNN sin PCA. En el resto de los tamaños se observan similitudes en las métricas que consideramos son lo suficiente buenas para justificar el uso de PCA.

Esto va en favor con nuestra hipótesis, los valores son muy similares. La explicación puede venir del hecho de que para este valor de alpha los resultados son muy buenos, teniendo poca perdida de información (esto fue probado mas arriba).

Experiencia al experimentar: Alpha igual a catorce: En la experimentación tomamos diferentes candidatos para nuestro alpha, ya que veíamos que la diferencia no parecía ser mucha. Por lo que probamos el experimento de diferentes tamaños con este alpha. Teniendo como resultado métricas muy distintas las cuales nunca superaban al algoritmo sin uso de PCA. Esto se puede ver en la fig.13.

IV. CONCLUSIONES:

En este trabajo estudiamos un conjunto de datos de 42000 imágenes manuscritas. Mediante el uso de métricas, logramos verificar la efectividad de nuestros algoritmos y comparamos las principales diferencias entre kNN y kNN+PCA. como el tiempo de cómputo y diferencias en las métricas. Además realizamos una comparación con distintos tamaños de imágenes. Esto nos permitió estudiar casos distintos, como el de una situación donde las clases estén desbalanceadas. Respecto a las hipótesis, concluimos que:

- H1: Pudimos ver que no necesariamente era así, ya que a partir de cierto k, esto deja de cumplirse. Es posible hallar un rango óptimo de k, particularmente en nuestro caso del 8 a 12.
- H2: Esto no es del todo cierto, ya que hay un punto en el que las métricas tienden a estabilizarse. Por lo que se podría hallar un valor de C no necesariamente grande y tenga unas métricas dentro del rango donde haya estabilidad.
- H3: A partir de cierto alpha, la información obtenida es apta para una adecuada métrica, ya que deja de haber mejoras significativas. Además, es importante recordar que el tiempo aumenta similarmente a una exponencial, por lo que hay que tener precaución con la elección del alpha.
- H4: La hipótesis resultó ser cierta, kNN en conjunto de PCA, resulto ser mucho mas rápida y con crecimiento menos empinado y además pudo igualar y en algunos casos supera las métricas medidas de kNN sin el uso de PCA En el caso del tiempo, podemos ver una gran diferencia. Para valores bajos esta es chica, pero a medida que aumentamos el tamaño del set, esta toma varios ordenes de magnitud de diferencia, siendo kNN+PCA mucho mas eficiente en este aspecto. Esto comprueba la hipótesis planteada.

REFERENCES

[1] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, 2009
[2] Notebook incluida en el git.
[3] PCA
[4] k-Fold
[5] H. F. Schantz, The History of OCR, Optical Character Recognition. Manchester Center, 1982.
[6] Y. LeCun, C. Cortes, and C. J. C. Burgess, "The mnist database of handwritten digits."
[7] N. S. Altman, "An introduction to kernel and nearest neighbor nonparametric regression," The American Statistician, vol. 46, pp. 175–185, 1992.
[8] I. T. Jolliffe, Principal Component Analysis. Springer, 2002.
[9] D. M. Allen, "The relationship between variable selection and data agumentation and a method for prediction," Technometrics, vol. 16, pp. 125–127, 1974.
[10] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009.

V. Apéndice:

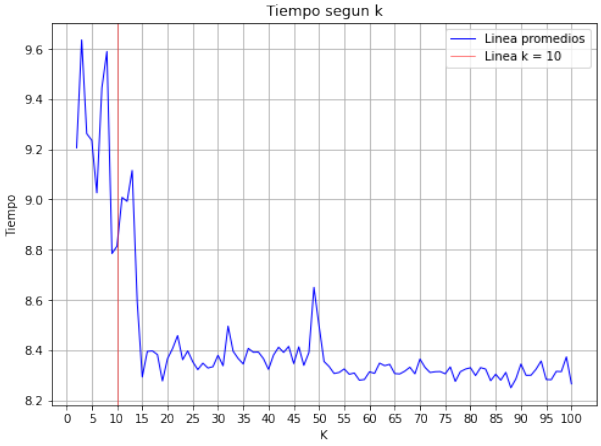

Fig. 9. Experimento donde se midió el tiempo de ejecución del programa (en segundos) con distintos k.

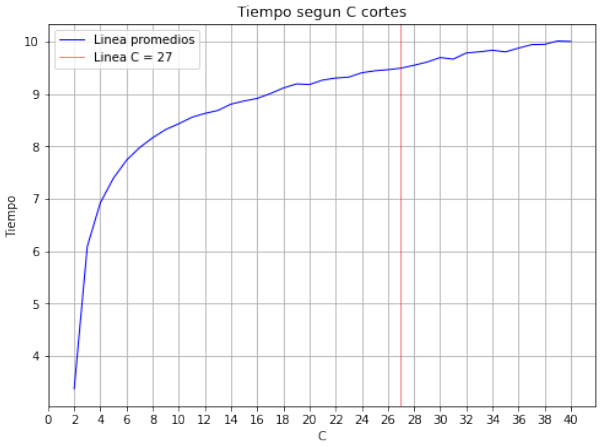

Fig. 10. Experimento donde se midió el tiempo de ejecución del programa (en segundos) con distintos C.

Fig. 11. Experimento donde se midió el tiempo de ejecución del programa (en segundos) con distintos alpha.

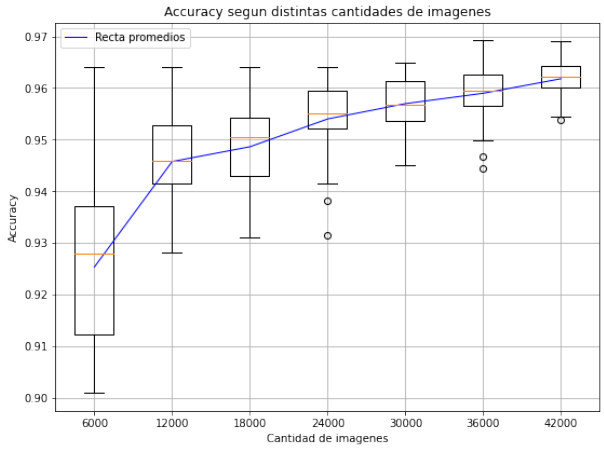

Fig. 12. Experimento donde se midió el accuracy de distintos tam. usando el algoritmo de kNN+PCA

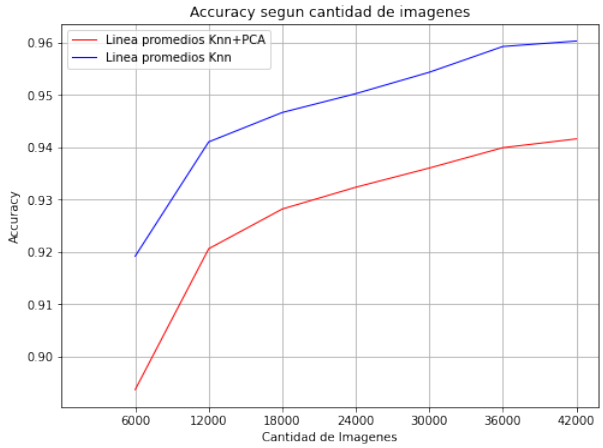

Fig. 13. Experimento donde se midió el accuracy de distintos tam. usando el algoritmo de KNN y kNN+PCA con alpha=14