DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

3 de diciembre de 2017

Métodos Numéricos
Segundo Cuatrimestre de 2017

Fuerte a Pachi

Integrante	LU	Correo electrónico
Tarrío, Ignacio	363/15	itarrio@dc.uba.ar
Szperling, Sebastián Ariel	763/15	sszperling@dc.uba.ar
Mena, Manuel	313/14	manuelmena1993@gmail.com
Frachtenberg Goldsmit, Kevin	247/14	kevinfra94@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Introducción

Este trabajo práctico tiene como objetivo el desarrollo y estudio de una herramienta que permita reconocer dígitos manuscritos en imágenes. El algoritmo capaz de llevar esto a cabo es uno de clasificación supervisado que fue entrenado con un lote de caracteres conocido, de modo que le sea posible reconocer otras instancias de esos caracteres aprendidos que no se encuentren en la base de datos de entrenamiento.

En cuanto al reconocimiento de dígitos, estos serán entre el 0 y el 9, y estarán plasmados en imágenes en escalas de grises.

1.1. Evaluación

Para el estudio de esta herramienta es necesaria la evaluación de los métodos y la correcta elección de sus parámetros. Una forma de evaluación es la estimación de la correctitud de la clasificación, para lo cual es necesita conocer previamente a qué dígito corresponde cada imagen. La forma de realizar esto 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, como por ejemplo el conocido caso de overfitting. Por lo tanto, se implementó la técnica de *K-fold cross validation* que resulta estadísticamente más robusta.

El resultado del algoritmo final fue medido con distintas métricas (Accuracy, Curvas de precisión/recall).

2. Desarrollo

2.1. kNN

El algorimo kNN (k Nearest Neighbours) se basa en el análisis de un conjunto de puntos del espacio para determinar a qué clase corresponde el nuevo objeto. En este caso cada imagen estará representada como un vector donde cada elemento es un píxel distinto de la misma. El conjunto de puntos elegido serán aquellas k imágenes que más cerca se encuentren de la imagen a clasificar. Lo que se hace es clasificar a la nueva imagen como la clase que mayor representantes tenga en este conjunto de puntos cercanos.

Este algoritmo puede ser sumamente costoso en cuanto al tiempo de cómputo, y si la dimensión de los puntos a clasificar es muy grande, hacer uso del mismo podría resultar impracticable. Es por esto que se implementó un método cuyo objetivo es preprocesar las imágenes para reducir la cantidad de dimensiones de las muestras, permitiendo a kNN trabajar con muestras con una cantidad de variables menor. Este método es conocido como PCA (Principal components analysis).

2.2. PCA

El método de análisis de componentes principales se encarga de cambiar de base el conjunto de datos de entrada para obtener una mejor representación de los datos, y además reduce la dimensión de cada elemento tanto como se desee. Al reducir la dimensión de un punto es claro que se pierde determinada información sobre el mismo, pero la particularidad de este método es que, como su nombre lo indica, se queda con las componentes más importantes, dejando de lado las que menos información aporten. De este modo, la información que se descarta es la que menor relevancia tiene, por lo que se lo considera una buena manera de reducir el espacio de la entrada.

Es importante aclarar que el método no solamente reduce la dimensión de los datos, sino que cambia la base de los mismos. Por lo que es necesario tener esto en cuenta al momento de trabajar con los mismos. Si se utiliza este método para reducir la entrada de kNN, es necesario cambiar a la misma base la imagen a clasificar, de lo contrario estarían comparándose elementos que viven en distintos espacios.

Procedimiento para el cambio de base:

1. Siendo X la matriz cuyas filas contienen las imágenes, se calcula $\mu = (x_1 + ... + x_n)/n$ el promedio de las imágenes, donde x_i es la i-ésima imagen.
2. Se crea la matriz X_μ que contiene en la i-ésima fila al vector $(x_i - \mu)^t$
3. Se calcula la matriz de covarianza $M = X_\mu^t X_\mu$
4. Se calculan los autovectores de M mediante el método de las potencias, con deflación. Cómo cada iteración en la que calculamos el autovector de la matriz en cuestión, este está asociado al autovalor de máximo módulo, los autovectores que habremos calculado se encontrarán ordenados por relevancia. De esta manera se calculan tan solo α autovectores, con $1 \leq \alpha \leq n$, siendo α la dimensión a la que se quiere reducir las imágenes.
5. Se contruye la matriz V con los autovectores calculados previamente, dispuestos como columnas. V es la matriz de cambio de base.
6. Por último, se reduce la dimensión de X cambiando su base. El resultado final es $V^t X^t$ que contiene la misma cantidad de imágenes pero expresadas en otra base, y en lugar de tener dimensión n, cada una tiene dimensión α.

Es interesante notar que una vez hecho el cambio de base de las imágenes, éstas representan a las imágenes pero si se trata de graficarlas se obtendrá algo muy distinto a lo que era anteriormente y no podrá visualizarse nada en concreto. Solo volviendo a la base original sería posible, pero ya se habrá perdido mucha información por lo que probablemente sea difícil encontrarlo útil.

2.3. Data augmentation

La performance de los métodos implementados y algoritmos similares está muy relacionada a la calidad y cantidad de datos. Por este motivo se nos ocurrió que aumentar la cantidad de datos podría ser una buena idea. Tenemos la hipótesis que al realizar un "Data Augmentation", agregando datos que podrían llegar a ser parámetros de entrada para clasificar(en nuestro caso dígitos manuscritos) obtendremos mejores resultados.

Para conseguir nuevos datos etiquetados, decidimos modificar las imágenes levemente para que estas sigan representando el mismo dígito.

Al ser dígitos manuscritos, los trazos no son perfectos, por lo que tratamos de encontrar alguna forma de aplicar transformaciones que nos generan ciertas irregularidades para imitar este comportamiento. Inspirandonos en este paper (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.160.8494&rep=rep1&type=pdf) usamos una técnica que se llama deformaciones elásticas, y las implementamos en matlab generando un desplazamiento aleatorio de cada pixel y después aplicando un filtro gausseano para suavizar este desplazamiento.

Deformación

Data: img, α, σ **Result:** La imagen manipulada

`/* Generamos 2 matrices de desplazamiento aleatorias */`

dx $\leftarrow$ rand(size(img))

dy $\leftarrow$ rand(size(img))

`/* Suavizamos con un filtro gausseano la aleatoriedad de la matriz */`

fdx $\leftarrow \alpha *$ imgaussfilt(dx, σ)

fdy $\leftarrow \alpha *$ imgaussfilt(dy, σ)

`/* Aplica el desplazamiento aleatorio usando la interpolacion de griddata */`

res $\leftarrow$ griddata(fdx, fdy, img)

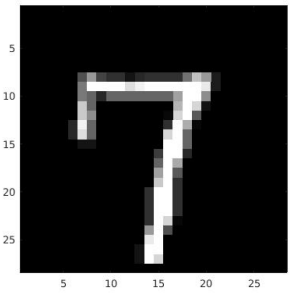

Figura 1: Original

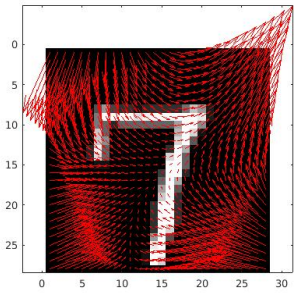

Figura 2: Vectores de desplazamiento

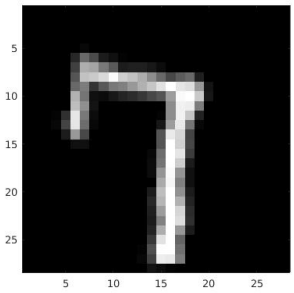

Figura 3: Resultado

Como se puede observar, el dígito resultante es distinto al original pero al ojo humano sigue siendo el mismo dígito. Con este método generamos un nuevo data-set aplicando esta función a cada imagen y así ahora tenemos el doble de datos 84000 dígitos.

Además de notar que los trazos no son perfectos, la orientación y rotación de los dígitos no es siempre la misma. Quisimos llevar al extremo aumentar los datos y aplicamos una segunda transformación a los dígitos. Para ello aplicamos una rotación aleatoria a cada dígito, usando la función de matlab para rotar imágenes imrotate, generamos un número aleatorio entre un cierto rango que le pasamos por parámetro como el ángulo. 30 grados nos pareció razonable ya que si se le aplica más hay dígitos que no se verían beneficiados.

En las imágenes se puede apreciar como el dígito 1 rotado es muy parecido al dígito 2, sin embargo es posible que el dígito 2 rotado pierda la esencia del dígito. Es por esta razón que optamos por usar rotaciones aleatorias.

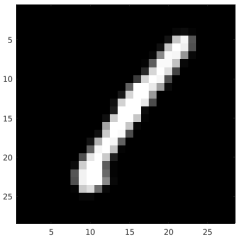

Figura 4: Dígito Original 1

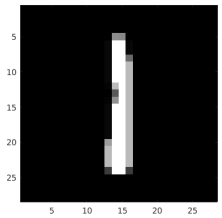

Figura 5: Dígito Original 2

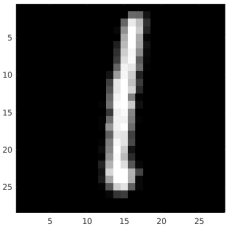

Figura 6: Dígito 1 rotado 30° sentido horario

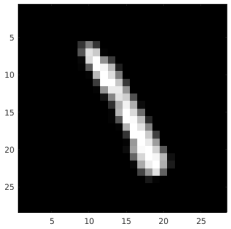

Figura 7: Dígito 2 rotado 30° sentido horario

3. Resultados

Uno de los objetivos de la experimentación de este trabajo práctico era encontrar los mejores parámetros para los métodos, es decir, los parámetros para los cuales los algoritmos proporcionaban mejores resultados. Para determinar la calidad de los resultados obtenidos (cuáles eran los mejores) se tuvo en cuenta distintas métricas, que ayudaron a determinar esto mismo:

- Accuracy
- F1-Score

Los resultados obtenidos fueron analizados en términos de estas métricas aplicando validación cruzada *K-fold* (sobre la cual se hablará a continuación) sobre la base de entrenamiento.

Los parámetros k (cantidad de vecinos en kNN) y α (dimensión a la cual se reduce cada imagen con PCA) fueron variándose como se expondrá en las páginas siguientes.

3.1. K-fold

La validación cruzada *K-fold* consiste en particionar la base de entrenamiento en K partes del mismo tamaño. Luego se realiza K iteraciones, cada una de ellas reteniendo uno de los conjuntos para validación y utilizando los restantes $K-1s$ para entrenamiento. Este método puede ser aleatorio, es decir, tomar las particiones sin cuidado alguno. Hicimos uso del método de MATLAB cvpartition, la cual tiene como define las particiones aleatoriamente de la cross-validation para un n y un K.

3.2. Tiempo de ejecución

Medimos 2 cosas en términos de los tiempos de ejecución: el tiempo que toma realizar PCA sobre el set de entrenamiento, y el tiempo en aplicar kNN sobre uno de los elementos (incluyendo el cambio de base en el caso de haber realizado PCA).

Para estas mediciones tomamos los sets de entrenamiento y prueba provistos por la cátedra, correspondientes a la competencia de Kaggle. También usamos para estas pruebas $k = 10$ (para kNN) y $alpha = 100$ (para PCA).

Los resultados fueron que, para el caso sin PCA, cada elemento de prueba tarda en promedio 500 ms en ser procesado. Consideramos que esto es relativamente lento, ya que al tratarse de 28,000 elementos, esto suma en total 3.89 horas, lo que resulta bastante prohibitivo y fue problemático a la hora de hacer experimentaciones.

En contraste, en el caso con PCA, cada elemento era procesado en aproximadamente 95ms, una mejora sustancial sobre todo considerando que esto incluye el cambio de base. Al aplicar sobre el set de prueba completo, esto demora alrededor de 45 minutos, una fracción del tiempo mencionado anteriormente.

Por último, cabe destacar que PCA en sí es un proceso costoso. Sin embargo, como nos esperábamos, el costo de PCA está compensado por las dimensiones de los sets utilizados, y aplicarlo sobre nuestro conjunto de entrenamiento demora en promedio 26 minutos, por lo que sumado a kNN, sigue representando apenas más del 25 % del tiempo de ejecución sin PCA.

Sin embargo, debe recordarse que el costo de PCA es elevado. Esto debe tenerse en cuenta junto con la precisión obtenida para sets de datos más pequeños, ya que podría no justificarse.

3.3. K de kNN

Para analizar el comportamiento al cambiar el K de kNN fijamos alfa de PCA en 2 valores 15 y 30. Corrimos experimentos para distintos estos K: 1, 2, 3, 4, 5, 7, 8, 8, 10, 15, 20, 30.

Nuestra hipótesis previo a correr los experimentos fue que para ks chicos, la posibilidad de que nuestro vecino más cercano sea un outlier representando otro dígito son más altas y para los ks grandes, pensamos que los dígitos que se encuentran en las fronteras, osea que se parecen a otros dígitos, se verían perjudicadas ya que por como funciona knn, pordía ocurrir que el más cercano sea, por ejemplo, 5 pero todos los siguientes sean 2. Por lo tanto, supusimos que los ks intermedios serían los que mejor funcionarían.

Habiendo corrido los experimentos, el gráfico de accuracy generado fue el siguiente:

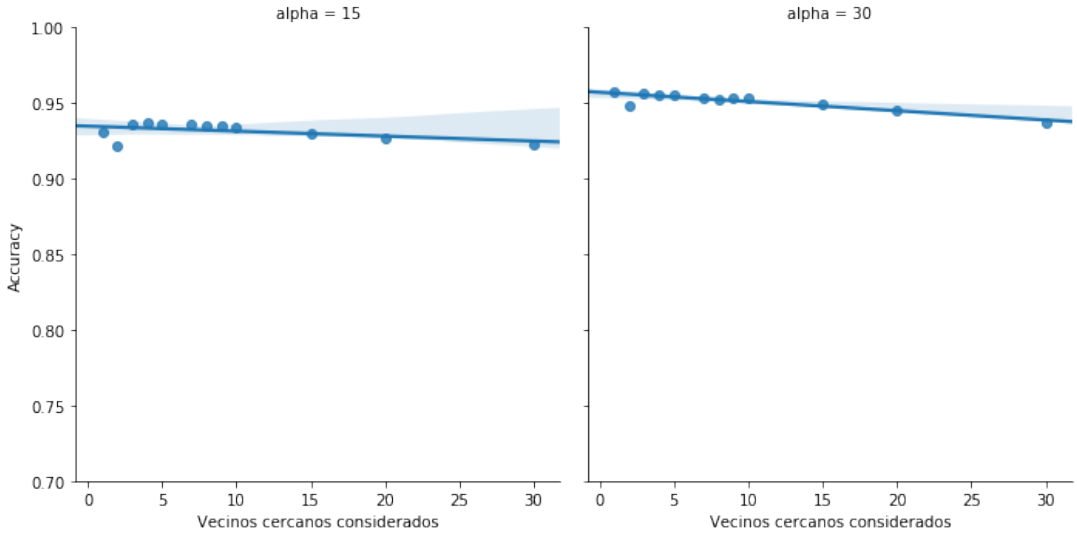

Figura 8: Accuracy por k de kNN con alfa de PCA fijo

Lo que se observó en este gráfico fue que el algoritmo es estable para los distintos ks y que las diferencias fueron poco significativas, pero pudimos contrastar con la hipótesis que lo que propusimos estaba en lo correcto.

El siguiente gráfico es una comparación del F1 score sobre los mismos resultados anteriores.

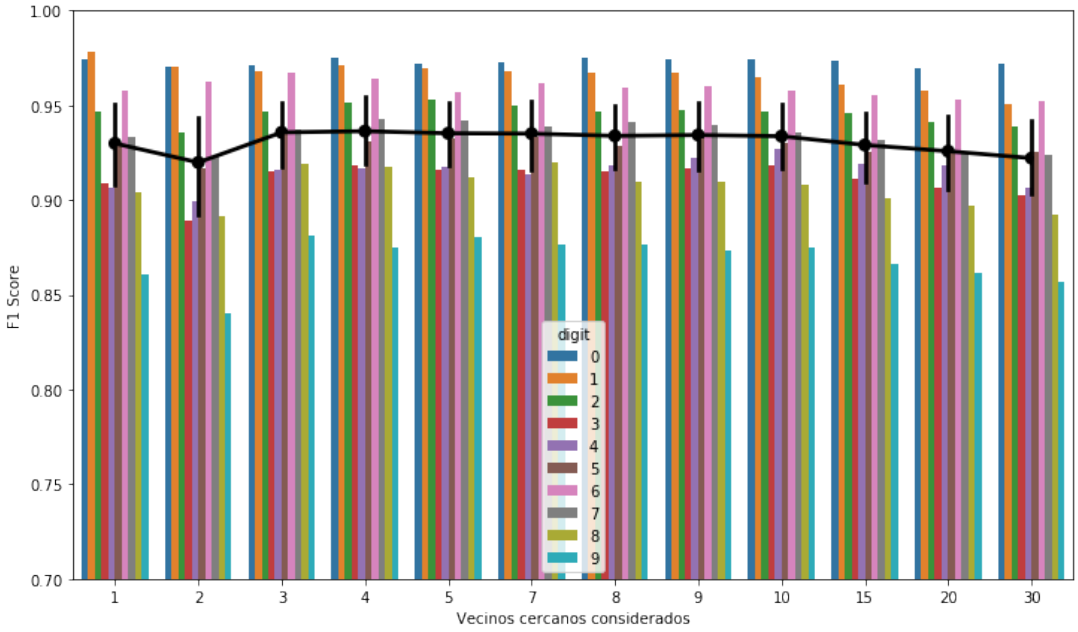

Figura 9: F1 score por k de kNN con alfa de PCA fijo en 15

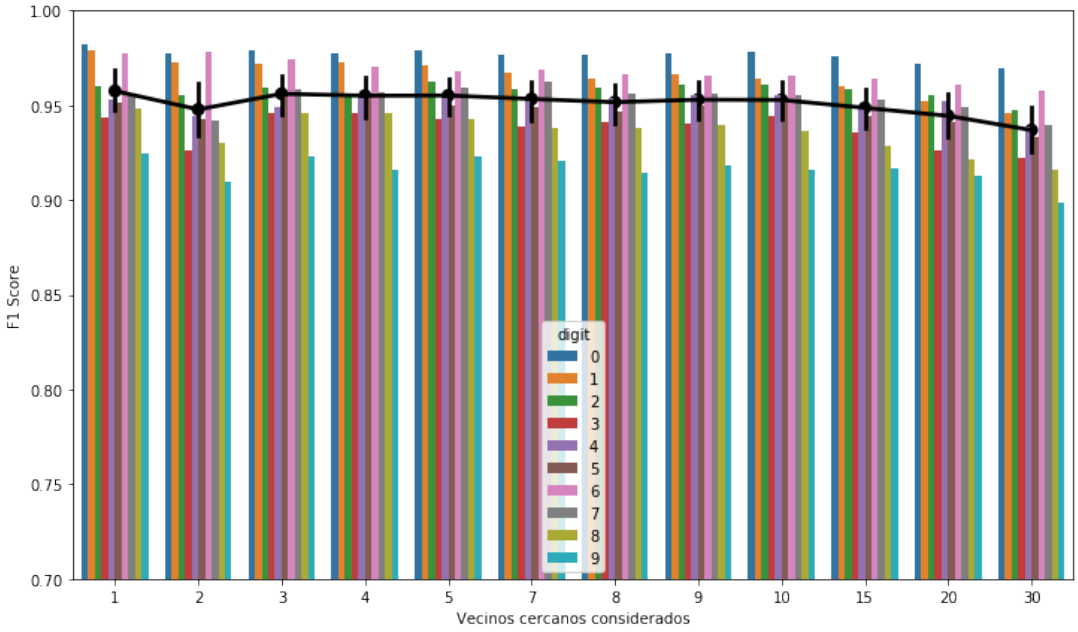

Figura 10: F1 score por k de kNN con alfa de PCA fijo en 30

Este gráfico nos mostró también que el algoritmo es estable para cada dígito y que para alfa 30 de PCA las diferencias entre los dígitos es menor. En la siguiente sección entraremos en detalle sobre la comparación que realizamos sobre el alfa de PCA.

3.4. Alfa de PCA

El otro parámetro que tratamos de optimizar fue el alfa de PCA, de una forma similar al K, fijamos este y fuimos cambiando el alfa, por suerte ya teníamos resultados sobre el K, así que decidimos fijarlo en 5. Los alfas con los que experimentamos fueron: 5, 10, 15, 20, 25, 30, 40, 50, 60, 80, 100, 150

Nuestra hipótesis para este experimento fue que si tomabamos un alfa bajo tendríamos bastante ruido y que para alfas grandes, al agregar tantas componentes, se daría menor importancia a los componentes mas relevantes por lo que perdería un poco de calidad de resultados ya que utilizar valores altos tiende a parecerse a sólo hacer kNN.

El siguiente gráfico muestra el accuracy para cada alfa evaluado.

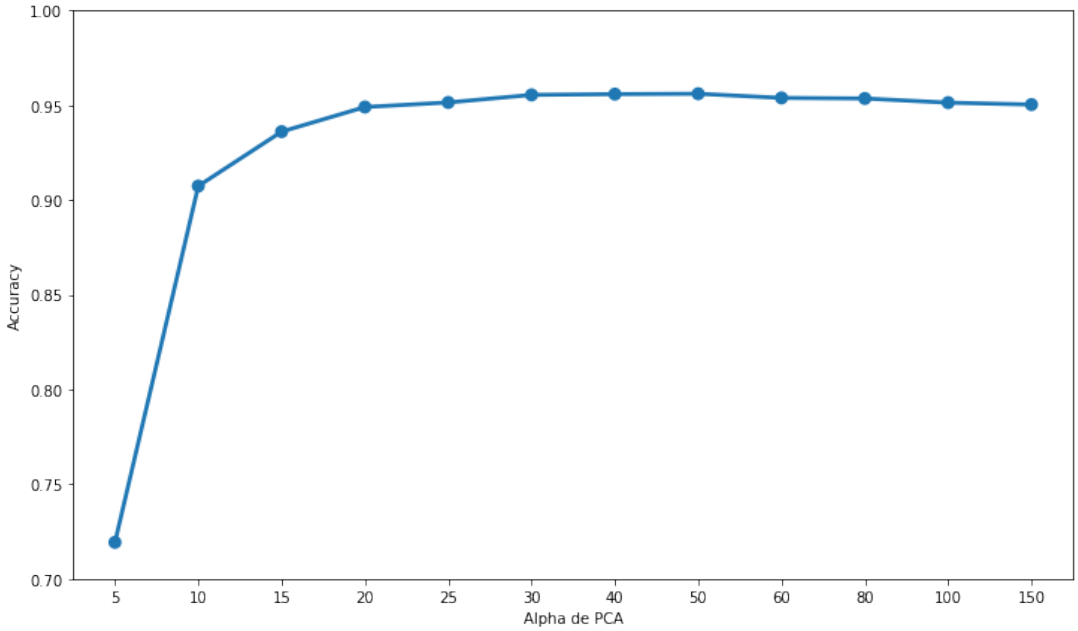

Figura 11: Accuracy por alfa de PCA con k de kNN fijo en 5

En este gráfico pudimos ver que los alfas que mejor funcionan de acuerdo al accuracy son los alfas entre 30 y 50 ya que a partir de ese momento es cuando el accuracy (lentamente) empieza a bajar.

El siguiente gráfico muestra F1 por dígito para cada alfa evaluado.

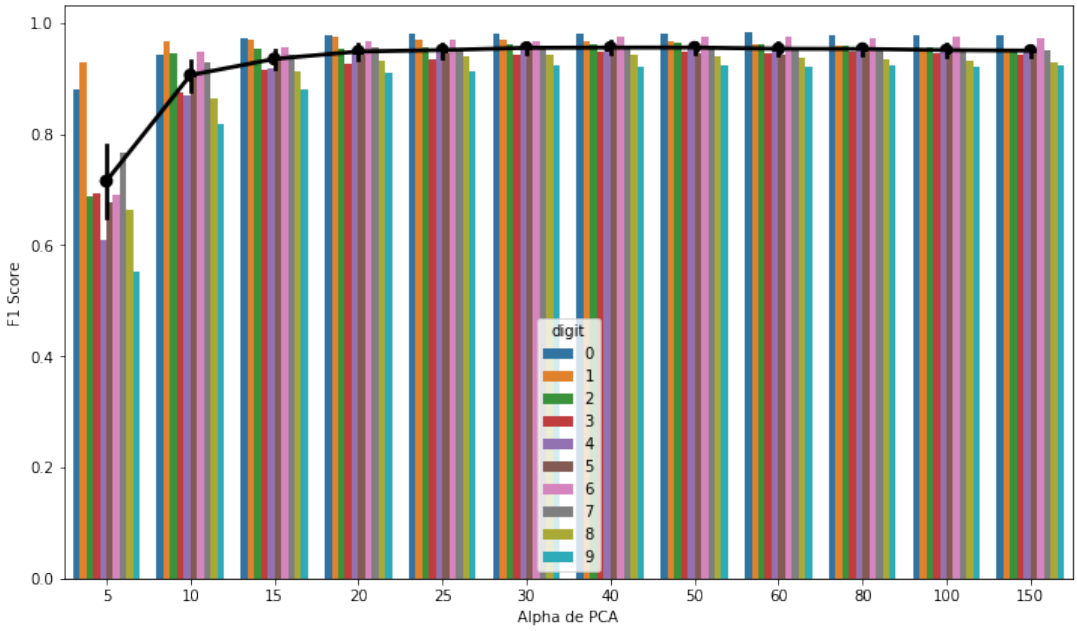

Figura 12: F1 score por alfa de PCA con k de kNN fijo en 5

En este caso se observó que para alfas muy bajos los resultados fueron bastante peores y que a medida que crece

el alfa, se estabilizan más los resultados. Sabemos que a mayor alfa, más lento anda el algoritmo, por lo que podemos concluir que no es bueno tomar este tipo de valores dada la pequeña diferencia de calidad que hay entre ellos. Por eso decidimos que 30 era un buen alfa.

En base a estos experimentos concluímos que nuestra hipótesis era correcta, aunque esperábamos que para valores como 150 ande peor que lo que realmente funciona.

3.5. Data augmentation

Antes de entrenar el algoritmo con todo el dataset quisimos ver cómo el modelo de transformaciones escalaba, para eso corrimos sobre un subconjunto del set de datos, a los mismos 10k digitos les aplicamos la transformación para obtener así un dataset de 20k. Como ya teníamos el K y el alfa analizados usamos 5 y 30 respectivamente. Los resultados del experimento fueron una accuracy de 0.9602 para las rotaciones y de 0.9715 para las deformaciones elásticas. Pudimos apreciar una gran mejora con las deformaciones elásticas, esto se puede deber a que las rotaciones hay casos en los que no queda tan real el dígito generado.

4. Conclusiones

En base a lo explicado en las secciones anteriores de este trabajo, pudimos concluir que los valores que mejor funcionan para nuestro algoritmo son 5 para kNN y 30 para el alfa de PCA. Más aún, podemos decir con confianza que el uso de análisis de componentes principales mejoró la calidad y el tiempo de ejecución de forma drástica para el set de datos provisto. No obstante, para que esto funcione de forma óptima se necesita un set de datos amplio como el utilizado para este trabajo. Además, encontramos que el método tiene un límite de precisión que no pudimos superar a pesar de realizar distintas técnicas y experimentaciones, si bien el mismo es elevado (casi 98 % de Accuracy).

En resumen, consideramos que este algoritmo es muy útil en casos donde una exactitud del 100 % no es indispensable, dada la calidad de los resultados y su sencilla implementación. Hacemos esta aclaración ya que este mismo algoritmo puede ser y es comunmente utilizado para múltiples problemas de clasificación, donde los datos de entrada son multidimensionales.

5. Apéndice

5.1. Apéndice I: detalles de implementación

Para poder soportar la posible necesidad de matrices más eficientes en memoria, creamos una clase Matrix con múltiples implementaciones. La misma tiene métodos para cada operación que consideramos necesaria, y algunas otras utilidades que utilizamos en el Trabajo Práctico 1. Por otro lado, implementamos una matriz completa (con vectores de la biblioteca estándar de C++), una matriz dispersa (usando mapas), y otras versiones más específicas.

Para evitar complicaciones con el manejo de punteros y el polimorfismo, utilizamos `std::shared_ptr` de C++11, que se encarga de borrar las matrices que no necesitamos por nosotros.

Sin embargo, luego nos dimos cuenta que los casos de uso de este TP no requieren varias versiones especificas como si las requería tal vez el TP anterior. La implementación sigue incluída, pero solo se usa la versión completa de la matriz (denominada `FullMatrix`).

Otro elemento que dejamos en la implementación pero no se utiliza fueron unos pseudo-iteradores para las imágenes que recorren archivos en lugar de vectores en memoria. El mismo no se usa porque es muy poco eficiente (ya que lee múltiples veces un archivo en disco), y luego de medir el consumo de memoria lo consideramos innecesario.

Por otro lado, decidimos abstraer el entrenamiento y el reconocimiento de las imágenes en una clase común, a modo de intercambiar implemetaciones rápidamente y por parámetros. Hoy en día nuestro TP solo cuenta con 2 implementaciones: kNN y kNN + PCA, pero podrían argegarse otras variaciones a futuro.

5.2. Apéndice II: código complementario en MATLAB

Además del código con el reconocimiento de dígitos implementado en C++, incluímos en la entrega algunos scripts de MATLAB que nos resultaron útiles durante la experimentación. Los mismos incluyen comentarios explicando su funcionalidad, pero todos están relacionados a los datos con los que expermientamos (con algunos métodos para probar incrementando el set de entrenamiento) y los resultados de dichos experimentos (validación cruzada, F1 score, etc).