DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Reconocimiento de digitos

2 de noviembre de 2020

Métodos Numéricos
2do Cuatrimestre de 2020

Integrante	LU	Correo electrónico
Olmedo, Dante	195/13	dante.10bit@gmail.com
Valls, Valentin Franco	449/19	valentinvalls@hotmail.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

Palabras Clave

Reconocimiento de dígitos, k vecinos más cercanos, Método de la potencia, Análisis de componentes principales

Índice

1. Resumen

En este trabajo implementamos y evaluamos un método simple de Optical Character Recognition para el reconocimiento de dígitos manuscritos y su digitalización. El método que usamos es conocido como k vecinos más cercanos, que se basa en predecir la categoría de una imagen utilizando las k imágenes que más se parecen. También hicimos uso del Análisis de componentes principales para mejorar su rendimiento. El objetivo que tuvimos fue analizar si este método es eficaz y ver si se obtienen diferencias notables con el uso de PCA. Como resultado de los experimentos se noto que efectivamente el uso de PCA mejora en varios aspectos nuestro método sin ninguna desventaja aparente. También se describió como buscar los parámetros a utilizar que traigan una mayor optimización, ya que la eficacia depende de valores como cuantos vecinos o cuantos componentes principales tengo en cuenta.

2. Introducción

Optical Character Recognition (OCR) [1] es ahora mismo una de las áreas de mayor interés para la computación, ya que la optimización de conversión automática de imágenes a formato digital implica un avance en la tecnología en general. Si bien esta área es muy amplia, en este trabajo nos enfocaremos en el reconocimiento de dígitos manuscritos y su translación al formato digital. Desarrollaremos una implementación que permita reconocer y diferenciar dígitos del 0 al 9 manuscritos y luego se evaluaremos su ejecución.

Para ello, utilizaremos un método de OCR simplificado, conocido como k Nearest Neighbors(kNN; k vecinos más cercanos) [2], con una reducción de dimensionalidad utilizando Principal Component Analysis (PCA; Análisis del componente principal) [3].

Para el entrenamiento de la implementación tomaremos MNIST [4], una base de imágenes de dígitos manuscritos. También utilizaremos MNIST para la evaluación, que a su vez aplicaremos la técnica de validación cruzada para corroborar los resultados, en particular el K-fold Cross Validation [2].

2.1. kNN

El método kNN se puede separar en dos momentos, entrenamiento y predicción.

El entrenamiento es la etapa inicial donde se recibimos un conjunto de imágenes ya categorizadas. Cada una de estas imágenes de $m \times m$ pixeles la convertimos en un vector donde cada elemento es un pixel de la imagen. De esta forma consideramos cada imagen como un punto euclídeo en un espacio m-dimensional y tiene asignado un dígito que lo representa.

En el momento de la predicción sobre una nueva imagen, convertimos la imagen recibida en un vector, de igual manera que en el entrenamiento, y luego compararemos la distancia euclídea de este nuevo punto con cada uno de los ya almacenados. Se recuerda que la distancia euclídea se esta definida como $d(x, y) = \sqrt{\sum_{i=1}^{m \times m} (x_i - y_i)^2}$.

Hecho esto, categorizaremos la nueva imagen dependiendo del dígito que predomine entre los k vecinos más cercanos. El valor de k es un factor importante para resultados de esta categorización, sobre todo en regiones del espacio euclídeo donde se acumulen puntos con diferentes dígitos asignados.

Lo que se nota de este método es que por cada predicción, se utiliza todo el conjunto de entrenamiento, aumentando su costo proporcionalmente al tamaño del conjunto de entrenamiento. Por ello es que se propone utilizar PCA para reducir las dimensiones del conjunto de entrenamiento.

2.2. PCA

Lo que se buscamos hacer con PCA es reducir las dimensiones del conjunto de entrenamiento condensando la información relevante en menos dimensiones. Intuitivamente, uno puede tener la idea de que los pixeles están correlacionados entre si, ya que los trazos en los manuscritos son continuos. Suena factible entonces intentar transformar al conjunto de entrenamiento a un conjunto ortogonal donde las dimensiones no estén correlacionadas.

El método PCA consiste en formar una matriz de covarianza del conjunto de entrenamiento, que al ser simétrica se puede diagonalizar. Definimos $X \in \mathbb{R}^{n \times m}$ como la matriz que representa a los datos de entrenamiento, $x_i \in \mathbb{R}^m$ la i-ésima imagen y fila de X y $\mu = (x_1 + ... + x_n)/n$. Entonces definimos M $\in \mathbb{R}^{n \times m}$ como la matriz que contiene en la i-ésima fila al vector $(x_i - \mu)^t/\sqrt{n-1}$. Por ultimo la matriz de covarianza de la muestra X se define como $C = X^t X$

Para diagonalizar C, buscaremos sus autovectores y autovalores con el método de la potencia para formar una matriz $T \in \mathbb{R}^{m \times m}$ que tendrá a los autovectores de C y una matriz $D \in \mathbb{R}^{m \times m}$ con los autovalores en la diagonal ordenados por valor absoluto descendiente que cumplan $C = TDT^t$ y llamaremos a los elementos de D como coeficientes principales.

Con T podemos transformar X, obteniendo una matriz $Y = XT$. Ahora como sabemos los coeficientes principales están ordenados, las columnas de Y estarán ordenadas por como maximizan la varianza del conjunto de datos mientras que las ultimas serán descartables. Una factor que queremos evaluar es como alpha$\in \mathbb{N}0$ <alpha$< m + 1$ hace que varíen los resultados de nuestra ejecución, ya que tendremos en cuanta solo las primeras alpha columnas de Y. En conclusión, dependiendo de que alpha tomemos, reduciremos la dimensión del conjunto de entrenamiento y reduciremos la redundancia de datos. Buscaremos un alpha que maximice la efectividad de resultado y minimice el costo cuando luego se aplique kNN.

2.3. K-fold Cross Validation

Para el momento de evaluar los resultados, introducimos la técnica de K-fold Cross Validation. La idea es que tenemos un conjunto finitos de imágenes ya categorizadas para entrenamiento, y para verificar si nuestros resultados son correctos, debemos tener imágenes ya categorizadas, por lo deberemos particionar las imágenes categorizadas entre conjunto de entrenamiento y de prueba. El problema es que si usamos siempre la misma partición, puede resultar en una incorrecta estimación de parámetros u overfitting y no podremos visualizar esto. El proceso que hacemos entonces es separar nuestra muestra en K subconjuntos de igual tamaño, tomamos K-1 conjuntos como un solo conjunto de entrenamiento y el ultimo subconjunto como conjunto de prueba. Repitiendo esto K-1 veces dejando cada vez un subconjunto diferente como conjunto de prueba, obtendremos unos resultados promedio menos propensos a errores.

3. Desarrollo

El desarrollo de este trabajo consistió por un lado en la implementación de kNN y PCA y por otro la experimentación para ver la calidad de la implementación y la búsqueda de parámetros óptimos para su funcionamiento.

3.1. Implementación kNN

La implementación de kNN se hizo generando una clase KNNClassifier en lenguaje c++ con un constructor que almacena k, una función fit y otra predict. Utilizamos la clase Matrix y Vector para almacenar el conjunto de imágenes recibido y poder hacer hacer operaciones sobre ella cómodamente.

Fit simplemente recibe el conjunto de entrenamiento y la categoría de cada elemento para almacenarlo en una matriz Data donde cada fila representa a una imagen y una matriz Tags de una sola columna donde cada fila tiene el dígito categoría de la respectiva fila de Data.

Luego para la predicción se recibe un conjunto de imágenes y por cada imagen a predecir tomada como un Vector v, se obtiene una nueva Matrix Substracción que es igual a Data pero a cada fila se le resta el vector v. Tomando el modulo de cada fila de Substracción tenemos la distancia de la imagen de v y cada imagen fila de Data, así que se tienen en cuenta las k-ésimas filas con menor modulo y tomamos el Tag predominante entre ellas para asignárselo a la imagen a predecir.

Entonces para hacer uso de kNN debemos crear un KNNClassifier con un k, usamos la función fit con datos ya categorizados y luego los datos que queremos categorizar. Notar que esta implementado para que el uso de PCA sea separable y pueda utilizarse o no antes de hacer el fit del KNN.

3.2. Implementación PCA

La implementación de PCA también se hizo en lenguaje c++ con una clase PCA, una implementación del método de las potencias para la obtención de autovalores. El constructor de PCA almacena el alpha que sera usado luego para saber cuantos autovalores obtener.

Las funciones de la clase PCA son fit y transform, la primera recibe una matriz, obtiene y almacena la matriz que la diagonaliza, y la segunda simplemente recibe una Matriz y transforma multiplicándola con la matriz almacenada en el fit.

3.3. Métricas de experimentación

Si bien tenemos ya que parámetros queremos estudiar, tenemos que definir que es lo que nos fijaremos con nuestros resultados. Nos fijaremos en el tiempo que tarda la ejecución, la Accuracy, el precision y el Recall [5] de nuestro resultados.

Si tenemos una muestra M de imágenes a predecir y tenemos nuestros resultados esperados y obtenidos, podemos definir los siguientes conceptos. Tomando T igual a la cantidad de resultados esperados iguales a obtenidos y F a

la cantidad de resultados esperados diferentes a obtenidos, podemos definir Accuracy como $\frac{T}{T+F}$. Por ejemplo si intentamos predecir 100 imágenes y solo 50 fueron categorizadas como era esperado, tendremos un Accuracy de 0.5. Si bien es algo muy útil y fácil de entender, no es una métrica perfecta. Tomando casos extremos donde nuestro M tenga una predominancia muy alta de un dígito en particular y nuestro algoritmo simplemente le asigne a todas las imágenes este mismo dígito, obtendremos un Accuracy alto que depende de M.

Si ahora analizamos por separado los resultados esperados y obtenidos con respecto de cada dígito, podemos agregar otras definiciones. Tomamos D como un dígito entre 0 y 9. Los casos donde esperemos D y obtenemos D serán TP(True positive), en cambio los casos donde obtenemos algo diferente a D serán FN(False negative). Los casos donde esperemos algo diferente a D y obtenemos algo diferente a D serán TN(True negative), en cambio los casos donde obtenemos D sera FP(False positive). Ahora precisión sera TP/TP+FP, esto representara la proporción de imágenes que fueron categorizadas como D y realmente eran D. Recall a su vez sera TP/TP+FN, esto representara la proporción de imágenes que eran D y fueron categorizadas como D.

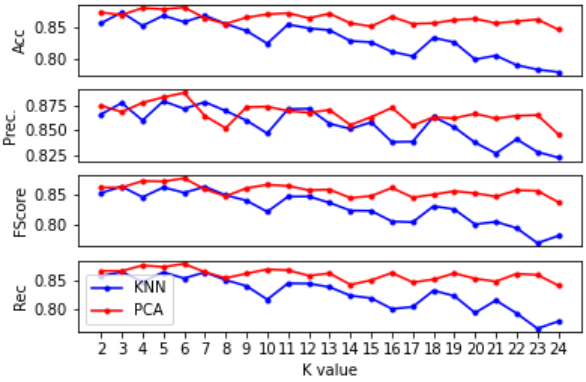
3.4. Experimentación

En este trabajo quisimos buscar un k para kNN, un alpha para PCA y un K para K-Fold validation que obtenga las mejores métricas posibles. Cabe remarcar que se quiso estudiar la diferencia de nuestras métricas con y sin PCA. Para ver nuestro k óptimo se experimento con diferentes k manteniendo alpha y K fijos, para alpha repetimos este proceso variando alpha manteniendo k y K fijo, y lo mismo para K. Como el conjunto de imágenes de MNIST tiene más de 40.000 elementos, para buscar los parámetros óptimos se opto por experimentar con un conjunto reducido y luego investigar como el tamaño de la muestra afectaba a nuestros resultados y al costo de ejecución.

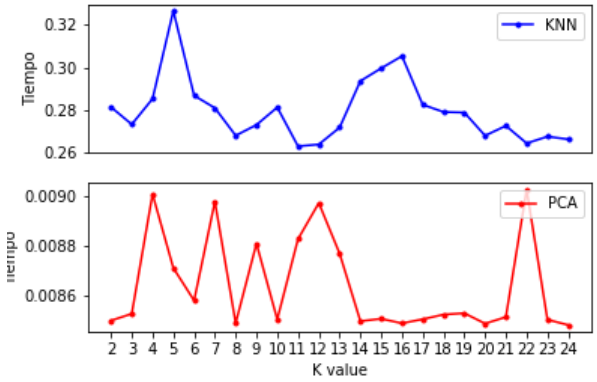
4. Resultados y discusión

4.1. Estudio de K

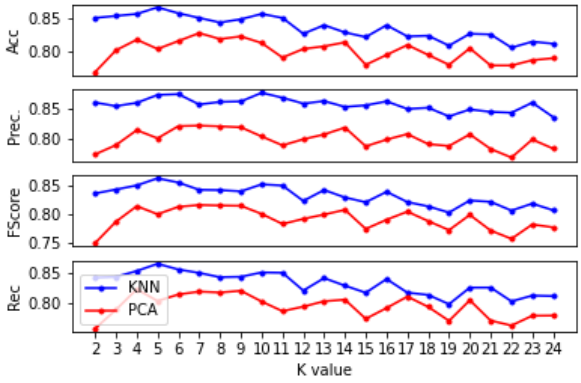
Inicialmente, decidimos experimentar con la cantidad de particiones que maneja K-fold. Se eligió como primer experimento porque la elección de un K adecuado afecta a todos los demás experimentos, y parecería afectar las probabilidades de todas las mediciones de manera constante, sin importar el resto de parámetros. Nuestra hipótesis a priori con respecto a la elección de K es que para K mayor a 1, aumentar la cantidad de particiones va a mejorar el accuracy de los experimentos pero para algún K en adelante va a empeorar, porque tomar un K demasiado grande daría mas imágenes para el conjunto de entrenamiento, pero achicaría el conjunto de test, y eso haría que los errores afecten mas el resultado promedio.



(a) Métricas variando K



(b) Tiempo variando K



(c) Tiempo variando K con otros parámetros

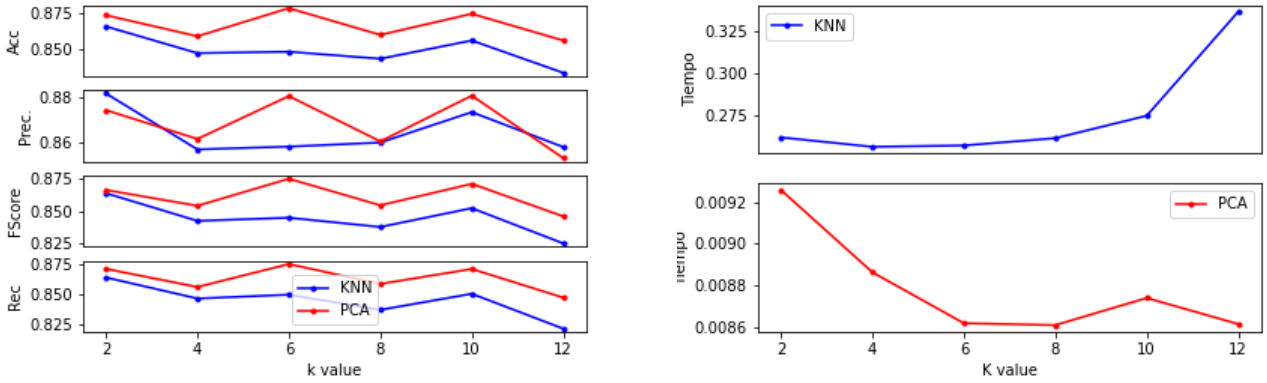
Figura 1: Experimentación variando K

Lo que podemos ver en la figura 1a apoya nuestra hipótesis. De 2 a 6 el accuracy sube hasta llegar al punto mas alto en el K=6, luego empieza a disminuir de manera no uniforme, pero empeora. Esto es mucho mas visible mirando el de Knn. La figura 1c corresponde a una ejecución que se hizo sin saber un valor correcto para alfa, por lo que nos da que Knn es mejor sin Pca, cosa que veremos luego que no es cierta dados los parámetros adecuados. Lo que se puede sacar de este primer resultado es el comportamiento general al variar K. Incluso con diferentes parámetros sucedió lo mismo, llegado un k bajo(en este caso el 5) el accuracy empezó a empeorar, y esto se mantuvo constante. Por los resultados vistos definimos que un K=6 es un valor aceptable para experimentar.

Para concluir con este análisis, del gráfico 1b podemos sacar 2 cosas. Primero, algo que se va a mantener constante a lo largo de los análisis, los tiempos de Knn son siempre peores que los de PCA, esto tiene sentido, dado que PCA reduce la dimensión de la matriz que ingresa a Knn, haciendo mucho mas rápido los cálculos posteriores. Lo segundo es que si bien el ruido puede hacer que varíen los tiempos de ejecución ligeramente no parece haber un patrón definido, por lo menos en nuestra ejecución, pero intenta acercarse a una función constante. Lo que nos planteamos como hipótesis en este caso es que K no modifica el tiempo que tarda una ejecución individual, pero modifica la dimensión de la matriz de entrada, porque cuanto mayor es K, mayor el conjunto de entrenamiento.

4.2. Estudio de k

La hipótesis inicial que se planteo respecto a k fue que mientras más vecinos se tengan en cuenta, mejores serán resultados serán pero solo hasta cierto punto. Ya luego de ese punto tener un k mayor nos traería peores resultados, se estarían agregando como si fuesen vecinos elementos que podrían muy alejados del punto a predecir. Por ejemplo un caso extremo donde k sea igual al tamaño del conjunto de entrenamiento, nuestra predicción solo dependerá de que categoría predomina en el total de imágenes. En términos de tiempo, tener un k más grande aumentaría nuestro costo, pero seria un costo lineal.



(a) Metricas variando k

(b) Tiempo variando k

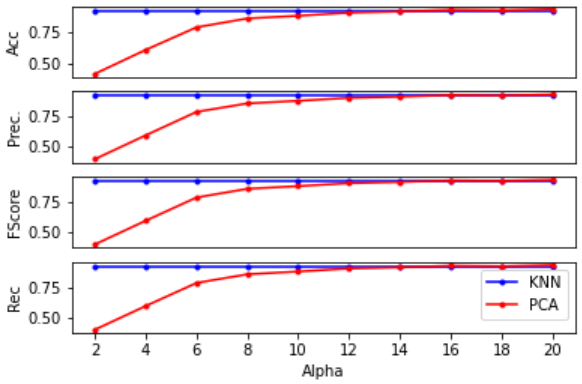
Figura 2: Experimentacion variando k

En la figura 2a tenemos varios puntos de análisis. Como primer observación, KNN es siempre peor sin PCA. Esto parece no ser así al ver la precision, para el último valor KNN da mejor sin PCA, esto podría deberse a que PCA concentra lo más importante de cada feature en los primeros vecinos, y al tomar más habrá más vecinos con información menos importante, mientras que knn puede agarrar más vecinos con el label correcto, y dar mayor cantidad de true positives gracias a ello. Otra cosa que podemos observar es un máximo para k=6, en 10 parece subir de nuevo el accuracy pero no es tan alto como el anterior máximo local, por lo que elegimos ese valor para correr la medición final.

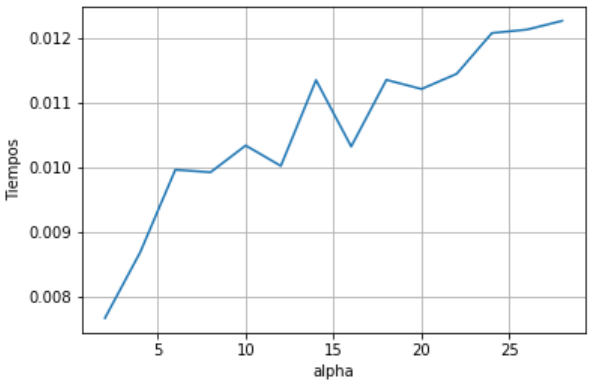
Respecto a los tiempos, vemos un aumento para knn al aumentar el k, esto es esperado, ya que la sección del código que lo usa es $O(k)$, pero esto no parece ir con la figura 2b, que parecería tener crecimiento cuadrático.

4.3. Estudio de alpha

Nuestra hipótesis inicial con respecto de alpha se baso en que las componentes principales están ordenadas, y mientras más componentes relevantes tome, mejores serán nuestros resultados. El punto importante es que luego habrán componentes despreciables que aumentaran el costo de ejecución pero no mejoraran drásticamente nuestros resultados.



(a) Metricas variando alpha



(b) Tiempo variando alpha

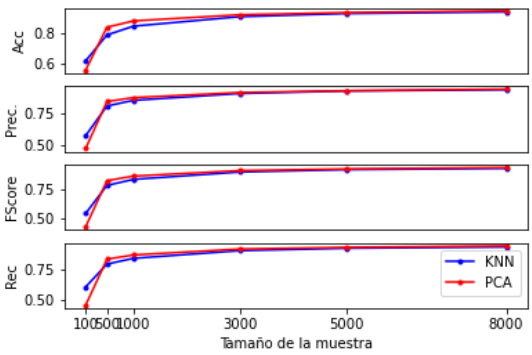
Figura 3: Experimentacion variando alpha

Para el experimento de la figura 3a se realizo la ejecución al variar solamente el alpha, es por esto que Knn es una constante, dado que no utiliza el parámetro. Se puede notar la mejora de PCA al aumentar el alpha, para eventualmente converger a un valor(0.95 aproximadamente) y también que dado un alpha lo suficientemente grande PCA es mejor que KNN puro.

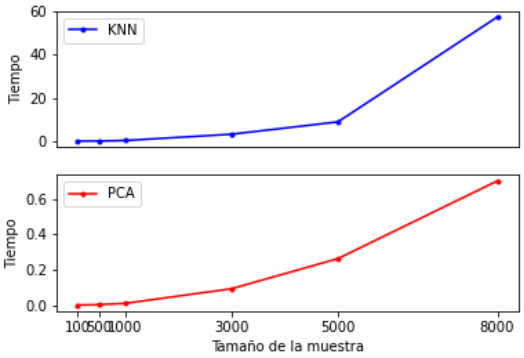
El valor que seleccionamos para nuestra ejecución completa es de alpha=18. Luego en la sección 4.5 se verá que esto no era suficiente para estimar el alpha.

4.4. Estudio del tamaño

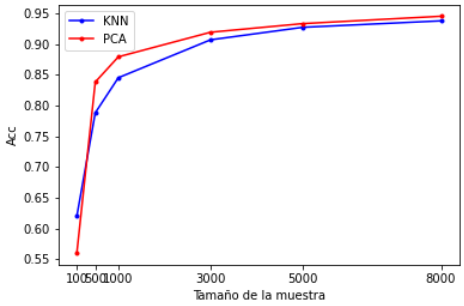
Nuestra hipótesis inicial con respecto al tamaño de entrenamiento fue que mientras más imágenes de entrenamiento tengamos, mejores resultados obtendremos, pero a su vez el tiempo de ejecución aumentará de manera cuadrática.



(a) Metricas variando el tamaño



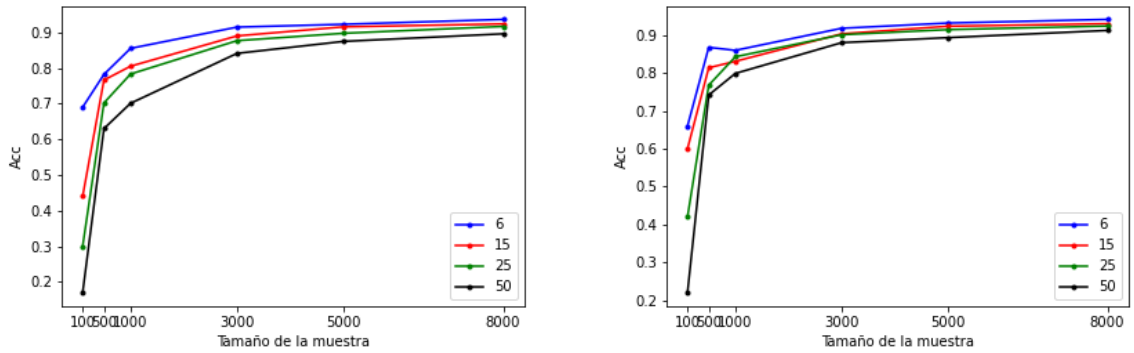
(b) Tiempo variando el tamaño



(c) Accuracy variando el tamaño

Figura 4: Experimentacion variando el tamaño

Empezamos hablando de la figura 4b, la cual indicaría que nuestra hipótesis a priori respecto al tiempo fue acertada.Ese gráfico corresponde a una función cuadrática. Como siempre, hay una diferencia de tiempos grande entre usar y no usar PCA(menos de un segundo para PCA con 1 minuto aproximadamente en caso contrario). Luego en las figuras 4a y 4c se nota como todas nuestras métricas toman una forma logarítmica con respecto al tamaño, lo que hace notar que pequeños conjuntos de entrenamiento no tendrán buenos resultados pero siempre mejorara con más muestras de entrenamiento tengamos. Cabe remarcar que llega un punto que, si bien aumenta la calidad de los resultados, no es proporcional al costo de ejecución.



(a) Accuracy con knn

(b) Accuracy con pca

Figura 5: Accuracy de varios k y varios tamaños

En estas figuras 5a y 5b buscamos analizar la relación de k con el tamaño del conjunto de entrenamiento. Se nota que cuando tenemos tamaños chicos hay una diferencia notable en el accuracy con diferentes k y mientras más pequeño es k mejor es el resultado. Con el aumento del tamaño, la diferencia disminuye, tendiendo al mismo numero manteniéndose siempre los valores menores de k con mejores resultados. A su vez se nota que con PCA la diferencia disminuye aun más rápido. Como trabajo futuro queda ver si hay algún punto donde con un tamaño de muestra mayor algún k mayor logre un mejor resultado.

4.5. Parámetros óptimos

Por los resultados de los estudios anteriores, finalmente se toman como óptimos los parámetros k=6, alpha=18, K=6. Hasta ahora la experimentación se hizo con un fragmento de la base completa de MNIST, pero una vez que se obtuvo estos parámetros óptimos se vio los resultados con MNIST completo.

```
Accuracy: 0.9649047619047618
Precision: 0.9655764321346788
Recall: 0.9643361435364487
F-Score: 0.9646930151299573
CPU times: user 57min 30s, sys: 1h 3min 23s, total: 2h 53s
Wall time: 2h 28s
```

(a) KNN

```
Accuracy: 0.9628571428571429
Precision: 0.9626419479131695
Recall: 0.9625381153314039
F-Score: 0.9625299295768864
CPU times: user 2min 23s, sys: 636 ms, total: 2min 23s
Wall time: 2min 23s
```

(b) PCA con alpha 18

```
Accuracy: 0.972547619047619
Precision: 0.9725522856057719
Recall: 0.9723032213753571
F-Score: 0.9723660543977217
CPU times: user 3min 26s, sys: 651 ms, total: 3min 26s
Wall time: 3min 26s
```

(c) PCA con alpha 30

Figura 6: Ejecuciones con la totalidad de los datos para knn y pca usando los parámetros obtenidos en la experimentacion

En la figura 6a se realizó una ejecución completa para knn y los parámetros previamente dichos, se logro una precision de 0.96 aproximadamente, que es lo mejor que encontramos hasta ahora en una medición. Esto era esperable pues en un experimento previo habíamos visto que el tamaño de la muestra aumenta el accuracy, y para esta ejecución usamos toda la muestra. Precision y recall nos dan valores muy similares, y acá puede verse el problema de ejecutar los experimentos con el set de datos completo. La ejecución tomo en total 2 horas y 28

minutos, lo cual si usasemos el set completo para cada experimento, sobretodo los que tienen varias iteraciones, nos daría con facilidad más de un día de ejecución.

Ahora veamos la figura 6b, esta ejecución es la de PCA con los parámetros vistos anteriormente, por lo que se esperaba un comportamiento similar a los experimentos antes realizados. Nos encontramos con que la ejecución nos dio un accuracy menor que KNN, cosa que con alpha=18 no venia sucediendo, ni es lo esperado. Ejecutar todo con PCA es relativamente barato por lo que decidimos hacer unas pruebas variando el alfa a uno más grande y encontramos que dado un alfa suficientemente grande volvíamos a tener una mejor accuracy de pca respecto a knn(Figura 6c). La ejecución de KNN nos sirve para comparar con esta nueva ejecución, pues alpha no modifica la ejecución de KNN, por lo que el resultado es el mismo. Esto deja ver que el alpha obtenido con los experimentos realizados en 4.3, si bien parecía un buen candidato, esta relacionado con el tamaño del conjunto y para conjuntos más grandes conviene tomar alpha mayores. Con el costo de un minuto más de corrida, cambiar alpha a 30 trae mejores resultados que kNN si utilizamos el conjunto de datos completo.

5. Conclusiones

Luego de un estudio del método kNN con y sin la aplicación de PCA, logramos demostrar la efectividad del método y el perfeccionamiento que logramos al aplicar PCA, se alcanzo un Accuracy de aproximadamente 0.97. Aplicar PCA implica una mejoría muy notable en el costo de ejecución, además de mejorar la calidad de nuestros resultados. Como parámetros óptimos que encontramos para nuestras experimentaciones con el uso de PCA fueron alpha=30, k=6 y K=6. Con respecto a estos parámetros notamos que siempre puede tomarse un alpha mayor al que elegimos si se desea obtener mejores resultados a costo de aumentar el tiempo. A diferencia de alpha, aumentar k y K no siempre traerá mejores resultados, sobretodo si tengo muestras pequeñas de entrenamiento, y aparte aumenta considerablemente el costo de ejecución sobretodo en muestras grandes.

Los siguientes experimentos quedan como experimentos futuros que pueden traer resultados interesantes.

- En la sección 4.4 se podría experimentar si hay algún tamaño mayor de los que experimentamos en lo que utilizar un k mayor trae mejores resultados.
- Utilizar otro set de datos distinto a MNIST
- Buscar casos borde para PCA y KNN, y analizar más a fondo con un set de datos muy chico, encontrar datos donde precision o recall den muy distinto a accuracy.

Referencias

[1] H. F. Schantz, “History of ocr, optical character recognition,” 1982.

[2] R. O. Duda, P. E. Hart, and D. G. Stork, “Pattern classification,” 2012.

[3] K. Esbensen and P. Geladi, “Principal component analysis,” 1987.

[4] Y. LeCun, C. Cortes, and C. J. C. Burges, “The mnist database of handwritten digits,”

[5] M. Sokolova and G. Lapalme, “A systematic analysis of perfomance measures for classification tasks,” 2009.