DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Nº 2

CSI:DC

Primer cuatrimestre de 2016 — Métodos Numéricos

Integrante	LU	Correo electrónico
Gonzalez, Juan Alberto	324/14	gonzalezjuan.ab@gmail.com
Rodriguez, Santiago	094/14	santi_rodri_94@hotmail.com
Sticco, Patricio Bernardo	337/14	pbsticco@hotmail.com
Walter, Nicolás	272/14	nicowalter25@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. Resumen

En el presente trabajo práctico nos proponemos estudiar técnicas utilizadas en el reconocimiento de dígitos. Para resolver este problema utilizaremos KNN. Como este algoritmo resulta lento para instancias grandes, pre-procesaremos las imágenes utilizando PCA o PLS. Realizaremos pruebas experimentales sobre estos métodos, variando sus parámetros de entrada, para comparar el rendimiento temporal y efectividad de cada uno.

Palabras clave: KNN, PLS, PCA, aprendizaje automático.

Ok.

2. Introducción Teórica

Falta describir el problema concreto y el contexto de aplicación

Contamos con una base de datos de n imágenes, que son utilizadas como instancias de entrenamiento, etiquetadas con el dígito correspondiente. Asumiremos que el etiquetado no contiene errores. El objetivo del trabajo consiste en utilizar la información de la base de datos para, dada una nueva imagen de un dígito sin etiquetar, determinar a cuál corresponde teniendo en cuenta factores de calidad y tiempo de ejecución requeridos.

Para el reconocimiento de imágenes utilizaremos kNN (k-nearest neighbors). En su versión más simple, este algoritmo considera a cada objeto de la base de entrenamiento como un punto en el espacio, para el cual se conoce a qué clase corresponde (en nuestro caso, qué dígito es). Luego, al obtener un nuevo objeto que se busca clasificar, simplemente se buscan los k vecinos más cercanos y se le asigna la clase que posea el mayor número de repeticiones dentro de ese subconjunto, es decir, la moda. Con este objetivo, podemos representar a cada imagen de nuestra base de datos como un vector $x_i \in \mathbb{R}^m$, con $i = 1, \dots, n$, y de forma análoga interpretar las imágenes a clasificar mediante este algoritmo.

$i = 1,\dots,n$.

El problema con kNN es que puede resultar muy costoso de computar a medida que aumenta la dimensión. Para mejorar los tiempos de cómputo, reduciremos las dimensiones de las imágenes utilizando PLS-DA (Partial least squares Discriminant Analysis); o bien, PCA (Principal component analysis). Además, al utilizar previamente estos algoritmos, la nueva imagen tendrá información más representativa.

Ok, y otros problemas.

3. Desarrollo

FALTA DESCRIBIR LA NOTACIÓN BASICA DEL PROBLEMA. CUÁNTOS DÍGITOS TENEMOS, COMO SE REPRESENTAN, ETC. ES NECESARIO A LO LARGO DEL TP.

3.1. Métodos numéricos usados

3.1.1. PCA

El objetivo detrás de este algoritmo es el de transformar un conjunto de datos con la finalidad de eliminar las redundancias e identificar los datos que aportan una mayor información. La manera en que realiza esto es mediante una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos donde la mayor varianza del conjunto de datos es capturada en el primer eje, llamado primer componente principal); la segunda varianza más grande es el segundo eje, y así sucesivamente. De esta manera, una vez aplicada la transformación lineal en las primeras α coordenadas se encuentran las variables que tienen menos covarianza, es decir, las que tienen menos dependencia y en las coordenadas restantes tenemos las variables más correlacionadas.

"variables" o "feature

Creo que es un problema de redacción. Está bien la idea de capturar la varianza.

En realidad, con la transformación en los nuevos ejes las variables están todas decorrelacionadas.

La matriz asociada a la transformación lineal que logra este cometido surge de calcular los autovectores asociados a los autovalores de mayor magnitud a Mx (la matriz de covarianza). Mx se obtiene de multiplicar X^tX,siendo X la matriz que tiene los datos en cada fila centrados con respeto a la media.

Ojo la notación, Mx es otra cosa que M_x.

respecto.

y escalados.

En nuestro caso, vamos a utilizar esta transformación lineal para cambiar de base nuestras imágenes y tratar de obtener así un clasificador mas eficiente.

Cuando se lo combine con knn.

Algoritmo 1: PCA

Datos: $X \in \mathbb{R}^{n \times m}, m \in \mathbb{R}^m$, μ_i que contiene el promedio de la columna i de X
Resultado: $\bar{X} \in \mathbb{R}^{n \times m}$, donde es la matriz de cambio de base

1 **para** $i \leftarrow 1$ **a** n **hacer**
2 $\quad X^{(i)} \leftarrow \left(X^{(i)} - \mu_i\right)/\sqrt{n-1}$
3 **fin**
4 $Mx \leftarrow X^tX$
5 $\bar{X} \leftarrow$ baseAutovectores(Mx)

Ok.

3.1.2. PLS-DA

PLS-DA, así cómo PCA, busca reorganizar el espacio de representación de los datos. A diferencia de PCA, donde la reorganización de datos se hace de una forma completamente no supervisada, en este caso se utilizan los valores de las clases para influenciar el traspaso al nuevo espacio de variables. Es decir, el método PLS-DA es un método supervisado.

OK.

Definimos X de la misma manera que en PCA. Es una matriz que tiene una muestra en cada fila, centradas con respecto a la media.Además, definimos Y como una matriz que en cada posición tiene la clase a la que pertenece la correspondiente muestra. Es decir, Y esta definido como: $Y \in \mathbb{R}^{n \times 10}$ y cumple $\forall i = 1, ..., n, \forall j = 1, ..., 10$

Acá están usando implícitamente que el problema y la notación son conocidos.

$$Y_{ij} = \begin{cases} 1 & \text{si la i-esima imagen tiene etiqueta j-1} \\ -1 & \text{caso contrario} \end{cases}$$

Vamos a buscar transformar las matrices a las formas $X = TP + E$ y $Y = UQ + F$ de tal manera de maximizar la covarianza entre las muestras y las clases en el nuevo espacio. Si estamos buscando un solo vector para realizar la transformación(t y u), vamos a utilizar el siguiente resultado. $Cov(t,u)^2 = Cov(Xw, Yc)^2 = \max\|r\| = \|s\| = 1$ Cov(Xr, Ys)^2.

El w que cumple esto es el autovector asociado al mayor autovalor de la matriz X^tYY^tX.

Algoritmo 2: PLS-DA

Datos: $X \in \mathbb{R}^{n\times m}, Y \in \mathbb{R}^{n\times 10}, \gamma$, *iter*
Resultado: $W \in \mathbb{R}^{m\times\gamma}$

para $i \leftarrow 1$ **a** n **hacer**
  $X^{(i)} \leftarrow \left(X^{(i)} - \mu_i\right)/\sqrt{n-1}$
fin
para $i \leftarrow 1$ **a** γ **hacer**
  $M_i \leftarrow X^tYY^tX$
  $w_i \leftarrow random()$
  $\lambda \leftarrow metodoPotencia(M_i, w_i, iter)$
  $w_i \leftarrow \frac{w_i}{||w_i||}$
  insertarEnColumna(W, i, w_i)
  $t_i \leftarrow Xw_i$
  $X \leftarrow X - t_it_i^tX$
  $Y \leftarrow Y - t_it_i^tY$
fin

OK.

3.1.3. *K*-fold cross validation

Al momento de experimentar necesitamos una métrica para estimar la eficiencia de nuestro algoritmo clasificador. Dichas métricas no deben verse sujetas al conjunto sobre el cual estamos aplicando el algoritmo. Es decir, necesitamos un algoritmo que sea capaz de predecir el resultado de una manera eficiente, independientemente del conjunto de *tests*. Ok, se entiende.

Ok, a esto me refería en el párrafo anterior.

Es importante conocer la eficiencia de nuestros algoritmos, para eso utilizamos distintas métricas. Si usáramos siempre los mismos conjuntos de train y de test, podríamos caer en 'overfitting', es decir, el algoritmo puede quedar ajustado a unas características muy específicas de los datos de entrenamiento que no tienen relación causal con los datos a testear. Ante esta problemática, al momento de experimentar vamos a utilizar *k-fold cross validation*, esta técnica es útil para evaluar los resultados de un análisis estadístico y garantizar la independencia entre la partición de entrenamiento y prueba.El método consiste en separar el conjunto de *training* en dos subconjuntos disjuntos. Uno será utilizado como *testing set* y vamos a validar cada uno de sus elementos (utilizando knn +pca, knn+ pls-da o simplemente knn) contra el otro subconjunto, denominado *training set*. Al realizar una sola partición del set de datos original se corre el riesgo de sobreajustar. Ojo la redacción, ya lo mencionaron antes.

Es por este motivo que debemos realizar k particiones en donde para cada partición, vali-

damos el testing set definido, con el training set. Con esto construimos un conjunto de entrenamiento para nuestro algoritmo en donde los resultados obtenidos van a ser independientes del conjunto sobre el cual se aplica, minimizando de este modo el riesgo de overfitting. Ok.

3.2. Algoritmos utilizados

Detalles de implementación?

Algoritmo 3: kNN

Datos: $X \in \mathbb{R}^{n \times m}, z^t \in \mathbb{R}^m$

1 $\text{argmin}_{i=1,....,n} \; ||X^{(i)} - z||$

Algoritmo 4: baseAutovectores Ojo! si solo computan \alpha, no necesariamente es una base de R^

Datos: $A \in \mathbb{R}^{m \times m}, \Lambda \in \mathbb{R}^{\alpha}$ ¿?

Resultado: $B \in \mathbb{R}^{m \times \alpha}$

1 **para** $i \leftarrow 1$ **a** α **hacer**
2 $\quad v \leftarrow random()$ Por qué random?
3 $\quad \lambda \leftarrow$ metodoPotencia(A, v, $iter$)
4 $\quad \Lambda_i \leftarrow \lambda$
5 $\quad$ insertarEnColumna(B, i, v)
6 $\quad$ Deflacion(A, λ, v)
7 **fin**

Algoritmo 5: metodoPotencia

Datos: $A \in \mathbb{R}^{m \times m}, x_0 \in \mathbb{R}^m, iter$

Resultado: λ

1 $v \leftarrow x_0$
2 **para** $i \leftarrow 1$ **a** $iter$ **hacer** Qué otros criterios consideraron como condición de corte? O solo tomaron cantidad de iteraciones?
3 $\quad v \leftarrow \frac{Av}{||Av||}$
4 **fin**
5 $\alpha \leftarrow \frac{v^t A v}{v^t v}$

Algoritmo 6: insertarEnColumna

Datos: $A \in \mathbb{R}^{m \times m}, i, v \in \mathbb{R}^m$

1 **para** $j \leftarrow 1$ **a** m **hacer**
2 $\quad aij \leftarrow vj$
3 **fin**

Algoritmo 7: Deflacion

Datos: $A \in \mathbb{R}^{m \times m}, \lambda, v \in \mathbb{R}^m$

1 $A \leftarrow A$ - λVV^t vv^t

4. Experimentación

4.1. *k*NN

4.1.1. Tiempo en función de la cantidad de vecinos

Presentación

En este experimento evaluaremos como se ve afectado el tiempo de cómputo del algoritmo al variar k.

Utilizaremos k-fold cross validation para evitar de este modo el riesgo de *overfitting*, con K=5 y K=15. Si están evaluando tiempo de cómputo, por qué es importa el overfitting?

Hipótesis

El tiempo de cómputo está directamente relacionado con la cantidad de vecinos a tener en cuenta.

Resultados

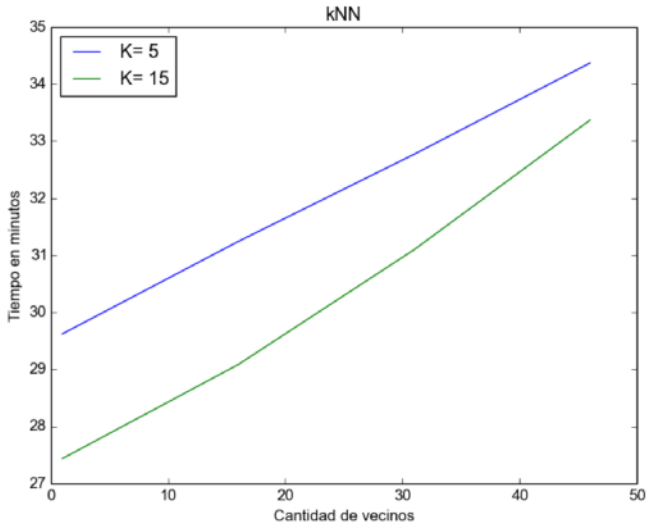


Figura 1: Gráfico del tiempo en función de vecinos, K = 5, K=15

Discusión

Como se puede observar en la figura anterior, el gráfico crece de manera lineal con respecto a la cantidad de vecinos. Esto se debe a que el método de kNN utiliza un algoritmo de orden O(n∗k) a la hora de encontrar la moda.

Ok, y que hay para decir respecto de la diferencia entre K=5 y K=15?
No queda claro que valores de k se están tomando.

4.1.2. Hit-rate en función de los vecinos

Presentación

En este experimento vamos a evaluar como varía el *Hit-rate* a medida que aumentamos la cantidad de vecinos a considerar.

Hipótesis

La cantidad de vecinos influencia directamente en el *Hit-rate* de forma tal que a mayor cantidad de vecinos a tener en cuanta mayor será el *Hit-rate* obtenido.

Datos utilizados

Para realizar este experimento utilizamos k fold cross validation, con K=5 y K=15.

Resultados

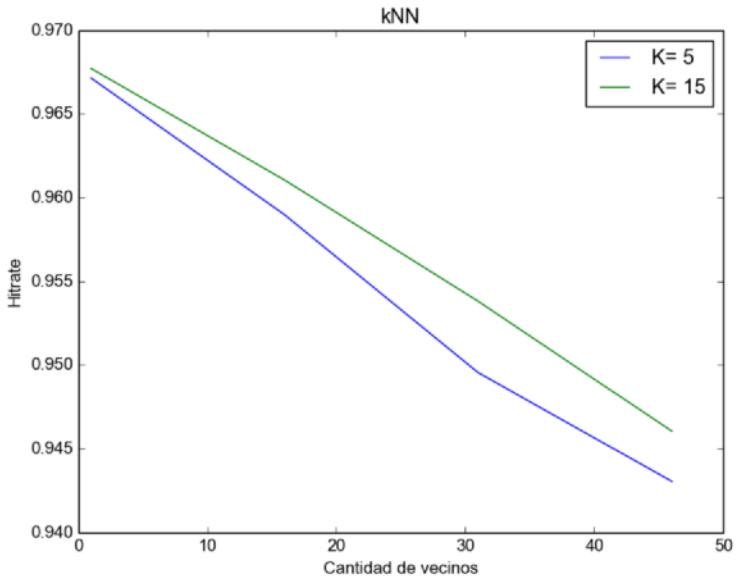


Figura 2: Gráfico de aciertos en función de vecinos, K = 5 y K=15

Discusión

Analizando los resultados obtenidos nos damos cuenta que los aciertos de nuestro algoritmo van disminuyendo a partir de cierta cantidad de vecinos contradiciendo así nuestra hipótesis.

Al examinar los resultados, podemos observar que alcanza máximo en el valor *k=1*. Considerando este hecho podríamos modificar el algoritmo para que nos devuelva siempre el elemento de menor distancia pero la exactitud de este algoritmo puede ser severamente degradada por la presencia de ruido o características irrelevantes, malogrando de este modo nuestras predicciones.

Ok, la idea es determinar esto experimentalmente

Si seguimos observando el gráfico podemos notar que al elegir *k=2* tenemos un hit-rate muy bajo. Esto podría deberse a que al tomar dos vecinos cuyas etiquetas sean distintas, la moda puede devolver cualquiera de las dos etiquetas y nada asegura que los resultados no se vean afectados por algún caso atípico.

Correcto.

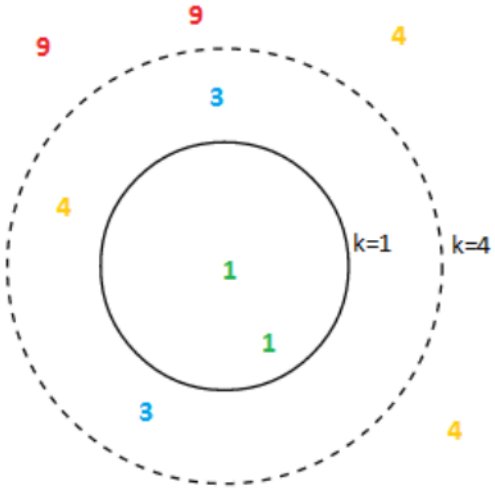


Figura 3: Ejemplo de una posible distribución de vecinos

No es conveniente tampoco considerar distancias demasiado grandes dado que en esos casos la moda se ve afectada por la cantidad de elementos considerados. Por ejemplo, en la imagen anterior al considerar 4 vecinos, la votación se ve afectada alterada devolviendo en ese caso un 3 cuando en realidad el elemento era un 1.

Una posible mejora ante esta problemática es considerar la distancia ponderada a la hora de la votación, es decir, otorgarles mayor *peso* a los elementos más cercanos en distancia mientras que a los elementos más lejanos otorgarles un peso menor. Una buena ponderación podría ser dividir cada distancia por su norma y de ese modo encontrar un máximo.

Bien.

Esta mejora es muy efectiva en muchos problemas prácticos. Es robusto ante los ruidos de datos y suficientemente efectivo en conjuntos de datos grandes. Queda propuesto, de este modo, esta mejora para futuras experimentaciones.

Conclusión

Concluimos, en base a los datos analizados anteriormente que no es conveniente utilizar valores demasiado chicos a la hora de tomar en cuenta los vecinos pues estos casos pueden verse alterados fácilmente ante la presencia de ruidos. Tampoco es conveniente considerar una cantidad de vecinos demasiado grande ya que en esos casos la votación se ve afectada directamente por la cantidad. Ok, alguna definición en particular en base a estos experimentos?

4.2. Técnica de pre-procesamiento alternativa

Presentación

Motivados por los resultados obtenidos en los experimentos anteriores y *la maldición de la dimensionalidad* que sufre kNN buscamos reducir el tiempo de cómputo utilizando alguna técnica de pre-procesamiento.

Analizando las imágenes de la base de datos provista por **Kaggle** observamos que la información más relevante a la hora de clasificarlas se encuentra en el centro de la misma mientras que en los bordes solamente se observan ceros correspondientes al color negro. Teniendo esto en cuenta vamos a buscar eliminar esta información irrelevante .

Hipótesis

Los bordes de las imágenes contienen información innecesaria a la hora de etiquetarlas. Por ello vamos a recortar dichos bordes reduciendo así la dimensión de las imágenes. La manera en que vamos a recortar la imágen es calculando el promedio del área utilizada para escribir. Suponemos que de esta manera vamos a reducir el tiempo de cómputo sin afectar de manera considerable las predicciones de nuestro algoritmo.

Promedio o max y min? Con el prom pueden perder información.

Datos utilizados

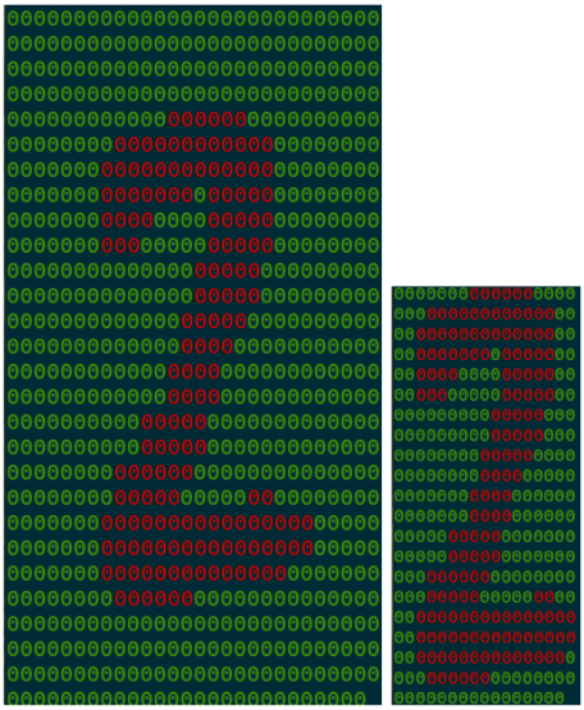
Para este experimento calculamos el promedio en donde se empieza a escribir y donde se termina de escribir. Para calcular dichos valores utilizamos las imágenes del training set y para uno de estos valores obtenemos la primera y última fila distinta de cero (donde terminan los bordes superior e inferior respectivamente). De igual manera calculamos el lugar donde termina el borde izquierdo y el borde derecho. Repetimos el procedimiento para las imágenes restantes y luego calculamos el promedio de estos valores.

Los resultados de este algoritmo aplicado a las 42000 imágenes fueron:

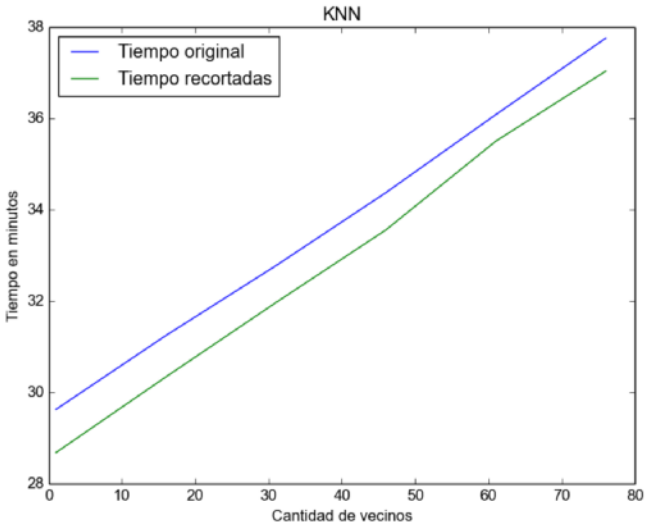
- Fila arriba: 4
- Fila abajo: 3
- Columna izquierda: 6
- Columna derecha: 5

Con estos resultados, la imagen después de ser recortada es un vector $v \in \mathbb{R}^{357}$.

Ok, interesante el experimento. La idea podría haber ido en el desarrollo.



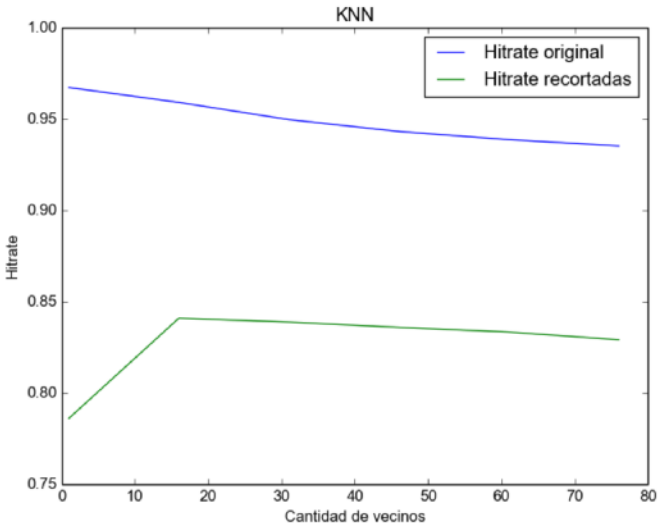
En primer lugar vamos a evaluar el tiempo de cómputo entre la imagen sin recortar y la imagen recortada. Esperando de este modo una mejora a favor de la imágen recortada.



OK.

Resultados cualitativos

Vamos a estudiar ahora la eficiencia del algoritmo clasificador con respecto al procesamiento previo realizado.



Qué valores de cantidad de vecinos tomaron en este experimento? Cuál es el valor de K usado para cross-validation?

Discusion

Analizando los gráficos anteriores podemos observar que el algoritmo kNN aplicado al set de imágenes recortada tiene un tiempo de ejecución menor que al aplicarlo sobre el set original acorde a los resultados esperados en la hipótesis. Esto se debe a que la dimensión de las imágenes después de recortar disminuye a 357. Reduciendo de este modo la dimensión en un 55 %.

Redacción. OK.

Luego de analizar los resultados obtenidos en la experimentación podemos apreciar una mejora en el tiempo de ejecución, como supusimos en nuestra hipótesis, pero el hit-rate disminuyó 15 % . Esto se debe a que perdemos información en el momento de recortar las imágenes, resultado que no esperábamos. Queda para futuras experimentaciones variar el tamaño a recortar y analizar cual seria el óptimo en términos del porcentaje de aciertos.

Ok, faltaría comprobar que se debe a eso (a la pérdida de información al recortar píxeles del dígito). Pregunta: cómo se les ocurre que podrían validarlo?

Conclusión

Luego de realizar este experimento, observando los datos arrojados, concluimos que esta técnica no resulta eficiente. Aunque se haya realizado una mejora en los tiempos de cómputo, la baja en la cantidad de aciertos es muy grande.

4.3. PCA

4.3.1. Eficiencia del método en función del alfa

Presentación

En esta sección vamos a tratar de analizar el comportamiento de nuestro clasificador usando la reducción de dimensión que nos brinda PCA. OK.

Datos utilizados

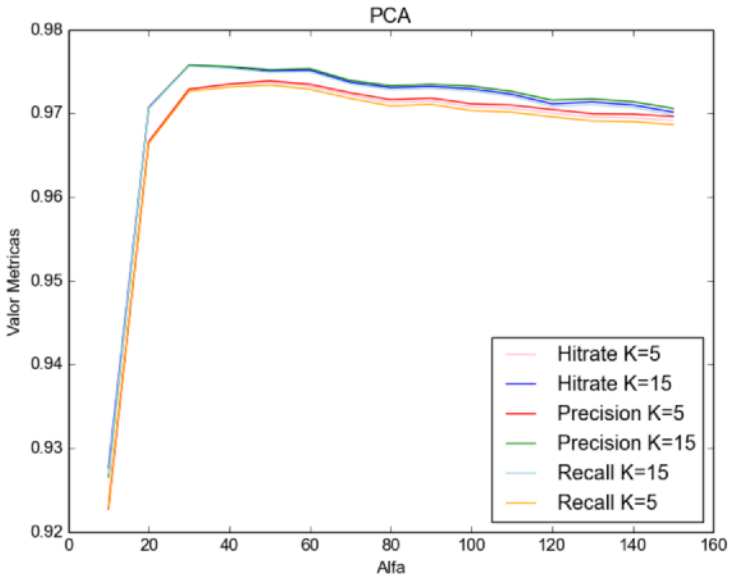
Para este experimento no nos va a interesar variar el k de kNN ya que eso sólo aumentaría la cantidad de variables independientes en nuestra experimentación. Por este motivo decidimos fijar la cantidad de vecinos k en 5, esta elección se desprende del resultado de la experimentación con Knn.
Nuevamente utilizaremos k-fold cross validation, con K=5 y K=15

Ok en aclarar el valor de k y K.

Hipótesis

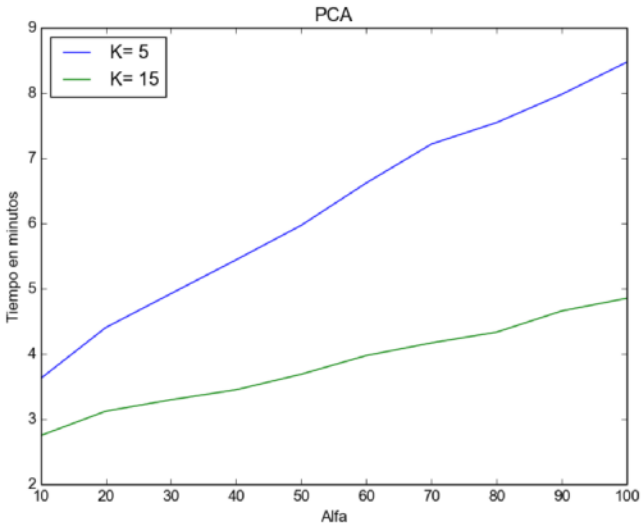
Nuestra hipótesis es que a mayor valor de *alpha* el tiempo de ejecución aumenta y la tasa de aciertos incrementa.

Resultados



Este gráfico no permite distinguir la diferencia en los valores, más allá de ver el rango en el que se encuentran.

Figura 4: Valores de las distintas métricas en función del Alfa



El tiempo considera el cómputo de la transformación característica?

Figura 5: Tiempo en función del Alfa

Discusión

En el gráfico anterior podemos observar que el *hit-rate* aumenta en función al tamaño del α hasta cierto punto, y luego comienza a disminuir. Esto se debe a que al tomar un alfa demasiado grande, dejaremos de usar los datos más relevantes. Al plantear la hipótesis no habíamos pensado en esa situación. Pero estábamos en lo correcto acerca del costo temporal del algoritmo.

En realidad, los datos más relevantes se usan. Se agregan ejes que tiene menor varianza.

4.4. PLS-DA

4.4.1. Eficiencia del método en función de γ

Presentación

En el siguiente experimento vamos a observar como se comporta el metodo PLSDA para diferentes valores de γ.

Hipótesis

Tanto el tiempo como la cantidad de aciertos incrementan cuando lo hace γ.

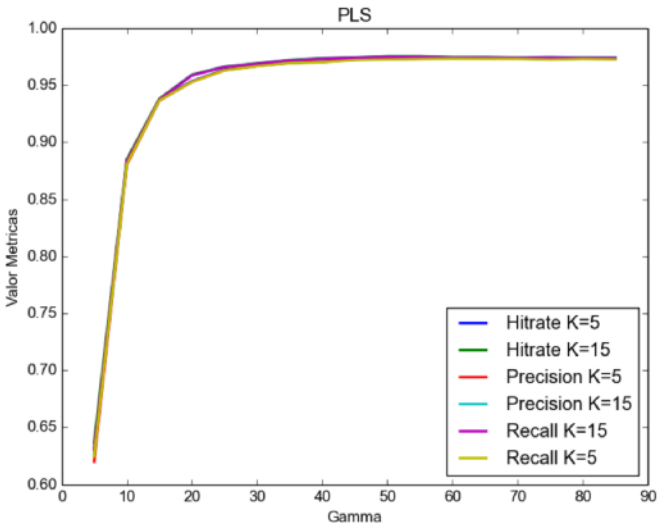
Datos utilizados

Para estos experimentos vamos a utilizar el algoritmo de kNN en conjunto con el algoritmo de PLS-DA .

Del mismo modo que en PCA vamos a fijar el k correspondiente al algoritmo de kNN en 5. También vamos a utilizar *k-fold cross validation* con K=5 y K=15.

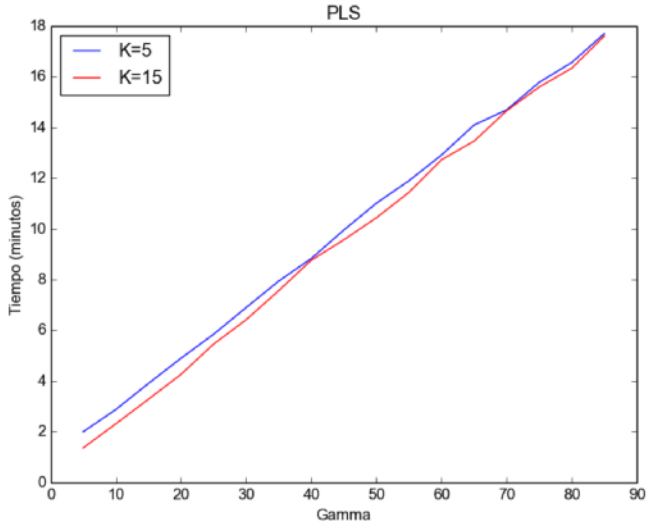
Resultados

Luego de realizado el experimento se obtuvieron los siguientes resultados:



Misma corrección que antes.
No se puede apreciar la magnitud
de cada métrica.

Figura 6: Valores de las distintas métricas en función del Gamma



Estan considerando kNN en esta
medición?

Figura 7: Tiempo en función del Gamma

Discusión

Como se puede observar en la figura 6, los aciertos del clasificador kNN aumentan a medida que la dimensión determinada por γ va en aumento. Esto se debe a que al aumentar la dimensión se agrega más información lo cual permite diferenciar entre etiquetas de una manera más precisa.

También aumenta el tiempo de cómputo considerablemente, consecuencia de aumentar el tamaño de las matrices con las que se trabaja. Esto es muy genérico.

4.5. Influencia de la cantidad de folds K sobre la eficiencia de los métodos implementados

Presentación

El objetivo de este experimento es estudiar como impacta la cantidad de folds en el costo temporal de los algoritmos utilizados previamente.

Datos utilizados

Utilizaremos dos valores distintos de K: 5 y 15. Estos fueron elegidos ya que creemos que al triplicar la cantidad de folds se podrá observar con facilidad sus diferencias, en caso de existir.

Hipótesis

Aca plantean mirar hitrate, no tiempos.

Mientras más grande sea el K más cantidad de aciertos tendrán nuestros algoritmos. Esto se deberá a que habrá una cantidad mayor de imágenes en el train y al tener una cantidad mayor los estimadores de mis parámetros van a ser mejores.

Resultados

Ver las figuras en los experimentos anteriores.

Discusión

En el caso de knn se puede observar que el tiempo de ejecución es mayor para K=5 que para K=15.

Esto se debe a que el tamaño del test set es igual al tamaño del train original dividido K. A mayor tamaño de conjunto de test, tardará más.

OK.

Podemos observar que el tiempo de cómputo del algoritmo PLS-DA no depende del tamaño de K ya que en el gráfico ambos se comportan de una manera similar para valores distintos de K. Esto se debe a que el algoritmo de PLS no tiene en cuenta el tamaño de la matriz X (que depende del tamaño del training set) más que en la primera iteración para γ para el cual calcula $M_i = X^t Y Y^t X$. Después de terminar esta primera operación el tamaño de X pasa a estar fijo en una matriz de $\mathbb{R}^{784x784}$ y al calcular los autovectores para hallar la matriz de cambio de base siempre lo hacemos sobre una matriz del mismo tamaño sin importar del K de donde provenga ni la iteración para γ en el que se encuentre.

Esto también sucede en PCA al calcular la matriz de covarianza.

4.6. Análisis de la eficacia de los métodos con respecto a K

Hipótesis

Mientras mayor sea K, el hitrate será más alto, ya que está relacionado con el tamaño del conjunto de entrenamiento.

Resultados

Ver figuras anteriores.

Discusión

La hipótesis es cierta para KNN. Al entrenar con más imágenes, aumenta su hitrate.

En los otros dos métodos, K no es tan importante, ya que aprovechan otras características de la base de entrenamiento, además de la cantidad de elementos. Por lo que no hay diferencias significativas en su eficacia.

Para PLSDA se puede apreciar que el *HitRate* es independiente al K, esto se puede deber a que en PLSDA representamos los datos de la manera X= TP + E y Y= UQ + F y en el algoritmo lo que tratamos de hacer es encontrar la máxima covarianza entre T y U. Si podemos encontrar una covarianza alta entre T y U, a partir de T podremos estimar U (pues están muy correlacionadas) y si logramos aproximar U, también aproximaremos Y, la matriz de clases. Una vez que aproximamos Y, etiquetaremos los valores de X. Recordando lo analizado previamente observamos que el valor de X no depende del training set. Entonces tampoco va a depender de K. Considerando este análisis no es sorprendente el método de PLS-DA no dependa del K elegido.

El hitrate, la precisión y el recall son muy parecidos siempre, esto se debe a que todos los datos están distribuidos en cantidades similares.

5. Conclusión

Observamos que es muy importante tener las herramientas matemáticas para el entendimiento de los problemas a resolver, para poder implementar métodos más eficientes. En este caso fueron PLS-DA y PCA. El hecho de explotar las características del problema, mejoró de forma realmente significativa los tiempos de cómputo sin perder eficacia en el reconocimiento.

Qué características explotaron?

Luego de observar los resultados, concluimos que el método más eficiente implementado resultó ser PCA, con alfa=60. Luego de ejecutar PCA, la cantidad de vecinas óptimas de knn es 5

No es cierto, primero fijaron k = 5 y despues analizaron PCA.