DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Re entrega Trabajo Práctico 2

"CSI:DC"

Metodos numericos
Primer Cuatrimestre de 2016

Integrante	LU	Correo electrónico
Ricardo Colombo	156/08	ricardogcolombo@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

Resumen

A lo largo de este trabajo abarcaremos distintas tecnicas y estrategias utilizadas en machine learninng intentando dar con la mas adecuada para obtener la mejor clasificacion de un conjunto de digitos manuscritos basandonos en la informacion mas relevante de cada una de ellas con el fin de poder realizar un reconocimiento de dichos digitos a partir de imagenes que los representan.

Palabras Claves— Machine Learning, Reconocimientos de digitos, K vecinos mas cercanos, Analisis de componentes principales. Regresion de minimos cuadrados

Índice

1. Introduccion Teorica

En el ambiente de la informatica en general se ha visto un incremento considerable en diferentes tecnicas que buscan tomar elementos del mundo como lo conocemos y darle una interpretación que hasta no hace mucho no era concebible para una maquina. Tales son los casos de reconocimiento de personas en imagenes con su posterior .etiquetado"(Facebook o Google Plus) o el reconocimiento de texto manuscrito en CAPTCHAs. El camino que muchas de estas compañías siguen es el de utilizar distintos metodos numericos para el reconocimiento de imagenes basado en un entrenamiento inicial realizado por los mismos usuarios de las redes sociales o sistema en cuestion. Hace años que Facebook nos deja .etiquetar.a nuestros amigos y familiares, esto se hace seleccionando una región de la imagen (cuadrado) donde usualmente esta la cabeza de la persona - con este accionar se mejor la taza de acierto considerablemente.

En el contexto de la materia de métodos numéricos, aplicaremos un enfoque similar al comentado anteriormente pero con un dominio menor (cantidad y complejidad de imagenes acotados). Esto no significa que nos estamos alejando de escenarios reales pero si que lo restringimos por practicidad. Dada una base de datos provista por la catedra con una gran cantidad de imágenes etiquetadas, intentaremos acertar cual es cual utilizando técnicas de Machine Learning - dichas tecnicas seran detalladas en los proximos parrafos.

Empezamos con la técnica **K vecinos más cercanos** (**kNN** dado las sigla en ingles k-nearest neighbor), a grandes rasgos podemos decir que teniendo un dominio de datos conocidos (etiquetados), se intenta de dar con aquellos que no tengan etiqueta utilizando para ello a los "vecinos". Se denominara "vecinos.a todo aquellos que estén más próximos a él en el espacio vectorial - esto se determina a través de la función de distancia. En dicha función las dimensiones del espacio están atadas al tamaño de la imagen, en nuestro caso R^{784} esto se debe a que las imágenes son de 28x28 y cada pixel representa una coordenada del vector con el que se representa la misma.

Siguiendo por esta línea continuaremos con dos método de reducción de dimensiones: Análisis de componentes principales (PCA) y Análisis discriminante con cuadrados mínimos parciales (PLS-DA). Estos métodos son similares en cuanto a que realizan una transformación característica pero difieren en cuanto a la información original que se utiliza para dicha transformación. A diferencia de **KNN**, estos métodos no son de clasificación, si no que sirven para hacer una clasificación de nuestros datos de entrada. Por consecuente utilizaremos una combinación de estos métodos **KNN+PCA** y **KNN+PLSDA** para cumplir con nuestro objetivo.

Finalmente, realizaremos un análisis de dichas técnicas. Cabe aclarar que dado que nuestro conjunto de datos es relativamente inferior comparado al cual compañías como Google y Facebook trabajan a diario, los tiempos de computos seran considerablemente inferiores pero agregando la dificultad del tipo de imagenes que vamos a estar reconociendo. Para esto tilizaremos un método denominado Cross validation. Este procedimiento consiste en, dado un conjunto de imágenes se realiza una partición con el fin de entrenar nuestro algoritmo. Se realizara una división del conjunto dado, la primera mitad será para entrenamiento y la segunda mitad para realizar pruebas con el fin de corroborar la predicción realizada.

2. Desarrollo

2.1. K vecinos mas cercanos

Como primera aproximacion para el problema de identificacion de digitos utilizaremos la de **K vecinos más cercanos** (**kNN** dada la sigla en ingles).La cual asume que las instancias son puntos en el espacio R^n.

Dada una instancia se definen a los vecinos mas cercanos como los elementos que tienen menor distancia a el, basandonos en la distancia euclediana. Cada instancia esta representado como un vector $x = (x_1, x_2....x_n)$, donde x_i es un atributo en especial, la distancia entre dos instancias esta dada por: $d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$ para luego quedarse con los k menores. Siendo estos los mas ¨parecidosä nuestra instancia.

Para realizar esto utilizaremos una funcion que nos dara la cantidad de apariciones que tiene una instancia dentro de un conjunto.

$$f(x) = argmax_{c \in C}(\sum_{i=1}^{n} \delta(v, f(x_i))$$

donde δ(a,b) devuelve 1 si a=b y 0 caso contrario.

2.1.1. Identificacion de vecinos

Para este trabajo en particular tomaremos las imágenes como vectores numéricos $x \in R^{784}$ debido a que cada imagen esta representada por una matriz de 28 pixeles de largo y 28 de alto(28 columnas y 28 filas). Decimos que dos imágenes son ¨parecidasde la misma forma que la definimos anteriormente, osea, si la norma dos entre ellas es pequeña.

Luego la idea del *knn* será tomar todas las imágenes etiquetadas, compararlas contra la nueva imagen a reconocer, ver cuales son las k imágenes cuya norma es la menor posible y, entre esos k vecinos, ver a que clase pertenecen.

Para los siguientes pseudocódigos será necesario asumir que todas las estructuras utilizadas almacenan datos de punto flotante a menos que se indique lo contrario, esto se indica agregando entre paréntesis el tipo de dato que almacena.

TP1 1 Vector KNN(matriz etiquetados, matriz sinEtiquietar,int cantidadVecinos)

```
1: vector etiquetas = vector(cant_filas(sinEtiquetar))
2: for 1 to cant_filas(sinEtiquetar) do
3:     etiquetas_i = encontrarEtiquetas(etiquetados,sinEtiquetar_i,cantidadVecinos)
4: end for
5: return etiquetas
```

A priori podemos suponer que al ir incrementando la cantidad de vecinos (k) menor va a ser la cantidad de aciertos, ya que se empiezan a mirar los elementos de menor prioridad de la cola, eso significa, que se cuentan primero las imágenes que más difieren y eso puede hacer que las chances de acertar el dígito correcto disminuyan. Ahondaremos mas en detalle en la siguiente sección, cuando pongamos a prueba cual es la mejor cantidad de vecinos para este algoritmo.
Al comienzo del desarrollo de los experimentos pensamos en diferentes maneras de mejorar el procesamiento de las imágenes, ya sea pasandolas a blanco y negro para no tener que lidiar con escala de grises o recortar los bordes de las imágenes, ya que en ellos no hay demasiada información útil (en todas las imágenes vale 0).

TP1 2 int encontrarEtiquetas(matriz etiquetados, vector incognito,int cantidadVecinos)

```
1: colaPrioridad(norma,etiqueta,vectorResultado) resultados
2: for 1 to size(incognito) do
3:    resParcial = restaVectores(etiquetados_i,incognita)
4:    colaPrioridad.push((norma(resParcial),etiqueta(etiquetados_i))
5: end for
6: vector numeros = vector(10)
7: while cantidadVecinos>0 & noesVacia(resultados) do
8:    int elemento =primero(resultados.etiqueta)
9:    numeros_elemento ++
10: end while
11: return  maximo(numeros)
```

Sin embargo, y mas allá de las mejoras que puedan realizarse sobre los datos en crudo, este algoritmo es muy sensible a la variabilidad de los datos. Un conjunto de datos con un cierto grado de dispersión entre las distintas clases de clasificación hace empeorar rápidamente los resultados.

2.2. Optimización mediante el analisis de componentes principales

El Análisis de Componentes Principales o PCA es un procedimiento probabilístico que utiliza una transformación lineal ortogonal de los datos para convertir un conjunto de variables, posiblemente correlacionadas, a un nuevo sistema de coordenadas conocidas estas como componentes principales tal que la mayor varianza de cualquier proyección de los datos queda ubicada como la primer coordenada (llamado el primer componente principal, aquella que explica la mayor varianza de los datos), la segunda mayor varianza en la segunda posición, etc. En este sentido, PCA calcula la base más significativa para expresar nuestros datos. Recordemos que una base es un conjunto de vectores linealmente independientes tal que en una combinación lineal, puede representar cualquier vector del sistema de coordenadas.
De esta manera entonces, será fácil quedarnos con los λ componentes principales que concentran la mayor varianza y quitar el resto. En la sección de experimentación, uno de los objetivos principales será buscar cual es el λ que concentra la mayor varianza de manera tal de optimizar el número de predicciones.

A fines prácticos, lo que haremos es, a partir de nuestra base de datos de elementos etiquetados, será construir la matriz de covarianza M de tal manera que en la coordenada M_ij se obtenga el valor de la covarianza del pixel i contra el píxel j. Luego, utilizando el método de la potencia, procederemos a calcular los primeros λ autovectores de esta matriz. Una vez obtenidos los autovectores, multiplicando cada elemento por los λ autovectores, obtendremos un nuevo set de datos. Sobre este set de datos, ahora aplicaremos el algoritmo KNN nuevamente y lo que esperamos ver es un mayor número de aciertos, ya que hemos quitado ruido del set de datos (mediante esta base que mencionamos al principio). Esto se suma a mejores tiempos de ejecución, ya que hemos reducido la dimensionalidad del problema.
Generalizando entonces, los supuestos de PCA son:

- Linealidad: La nueva base es una combinación lineal de la base original.
- Media y Varianza son estadísticos importantes: PCA asume que estos estadísticos describen la distribución de los datos sobre el eje.
- Varianza alta tiene una dinámica importante: Varianza alta significa señal. Baja varianza significa ruido.

- Las componentes son ortonormales.

Si algunas de estas características no es apropiada, PCA podría producir resultados pobres. Un hecho importante que debemos recordar: PCA devuelve una nueva base que es una combinación lineal de la base original, limitando el número de posibles bases que puedan ser encontradas.

2.2.1. Algoritmos para Optimización con PCA

TP1 3 void PCA(matriz etiquetados, matriz sinetiquetar,int cantidadAutovectores)

```
1: matriz covarianza = obtenerCovarianza(etiquetados,medias(etiquetados))
2: vector(vector) autovectores
3: for 1 to cantidadAutovectores do
4:     vector autovector=metodoDeLasPotencias(covarianza)
5:     agregar(autovectores,autovector)
6:     double lamda = encontrarAutovalor(autovector,covarianza)
7:     multiplicarXEscalar(auovector,lamda)
8:     restaMatrizVector(covarianza,auovector,lamda)
9: end for
```

TP1 4 matriz obtenerCovarianza(matriz entrada,vector medias)

```
1: matriz covarianza, vector nuevo
2: for i=1 to size(medias) do
3: for j=1 to cant filas(entrada) do
4:     nuevoVector_j = entrada_(j,i) medias_i
5: end for
6: agregar(covarianza,nuevoVector)
7: end for
8: for i=1 to cant filas(entrada) do
9: for k=1 to cant filas(entrada) do
10: covarianza_i = multiplicarVectorEscalar(covarianza_k, cantidadFilas(entrada))
11: end for
12: end for
13:
14: return covarianza
```

Además implementamos los métodos auxiliares para el metodod e las potencias como para obtener el vector de medias.

TP1 5 Vector metodoDeLasPotencias(matriz covarianza,cantIteraciones)

```
1: vector vectorInicial= vector(cant filas(covarianza))
2: for 1 to cantIteraciones do
3:     vector nuevo = multiplicar(covarianza,vectorInicial)
4:     multiplicarEscalar(nuevo,1/norma(nuevo))
5:     vectorInicial = nuevo
6: end for
7:
8: return vectorInicial
```

TP1 6 Vector medias(matriz entrada)

```
1: for i=1 to cantColumnas(entrada) do
2:     suma = 0
3:     for j=1 to cant columnas(entrada) do
4:         suma += entrada_{i,j}
5: end for
6: medias_i = suma/cantFilas(entrada)
7: end for
8:
9: return medias
```

2.3. Optimización mediante la regresion por Cuadrados Minimos Parciales

Por ultimo analizaremos la otra transformacion, que si bien es similar al anterior (**PCA**), la principal distincion entre ambos es que este utiliza las clases para influenciar el traspaso al nuevo espacio de variables, conviritiendolo asi en un metodo supervisado. El metodo PLS-DA consiste en una regresion clasica PLS, que permite el manejo de datos multi categoricos a diferencia de PLS que es bilineal.

Si estuvieramos definiendo el PLS clasico deberiamos definir X de igual manera que se hizo en el algoritmo anterior, osea una matriz donde tenemos en cada fila una muestra, centradas respecto de la media, queremos buscar transformar las matrices con el fin de maximizar la covarianca entre las muestras y las clases en el nuevo espacio quedandonos de esta manera.

$$X=TP+E$$
$$Y=UQ+F$$

Si definimos como t y u los vectores de las matrices de transformacion T y U, se busca maximizar la covarianza.

$Cov(t,u)^2 = Cov(Xw,Yc)^2 = max_{||r||=||s||=1} Cov(Xr,Ys)$

Donde el w que cumple esto es el autovector asociado al mayor autovalor de la matrix $\mathbf{X^tYY^tX}$.

Una vez realizado esto utilizaremos en metodo de deflasion para quitar este autovector para asi obtener el primero de los γ mayores autovectores mediante un esquema iterativo.

Definimos Y como la matriz resultante de tomar la matriz Y' $\in R^{n10}$ como una matriz que tiene un 1 en la posicion Y'$_{i,j} silamuestraidelabasedeentrenamientocorrespondealdıgitoj1, y{-}1enelcasocontrario, sustraere$

2.3.1. Algoritmos para Optimización con PLS-DA

TP1 7 void PLSDA(matriz etiquetados, matriz sinetiquetar,int γ)

1: matriz X = matX(etiquetados)
2: matriz Xt = traspuesta(X)
3: matriz Y = preY(etiquetados)
4: matriz Yt = traspuesta(Y)
5: for 1 to γ do
6: vector w=metodoDeLasPotencias($X * Y * Yt * Xt$)
7: agregar(autovectores,w)
8: t$_i$ = $multiplcarMatrizVector(X, normalizar(w)$
9: X = X - t$_i$ * $traspuesta(t_i) * X$
10: Y = Y - t$_i$ * $traspuesta(t_i) * Y$
11: end for
12:
13: **return** maximo(numeros)

TP1 8 Matriz X(matriz etiquetados,vector medias, int n)

```
1: matriz X = nueva MatrizCuadrada(n)
2: para i de 1 a n
3:    para j de 1 a size(etiquetados)
4:       X(i,j)=etiquetados[j].vect[i] - medias[i])  √(n − 1)
6: return X
```

TP1 9 Matriz preY(matriz etiquetados)

```
1: matriz Y = nueva Matriz(1,10)
2: para i de 1 a n
3:    para j de 1 a 10
4:       si etiquetados[i].label-1 = j-1
5:          Y[i][j]=1
6:       si no
7:          Y[i][j]=-1
8:
9: return Y
```

2.3.2. Cross-Validation

Para medir la precisión de nuestros resultados utilizamos la metodología de cross-validation. Esta consiste en tomar nuestra base de datos de entrenamiento y dividirla en k bloques. En una primera iteración se toma un bloque para testear y los bloques restantes para entrenar a nuestro modelo, observando los resultados obtenidos. En la siguiente, se toma el segundo bloque para testear y los restantes como dataset de entrenamiento. La metodología se repite k veces hasta iterar todo el conjunto de datos. Finalmente, se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado de error y poder evaluar la performance del método de entrenamiento.

Esta técnica, que es una mejora de la técnica de holdout donde simplemente se divide el set de datos en dos conjuntos (uno para entrenamiento y otro para testing), trata de garantizar que los resultados obtenidos sean independientes de la partición de datos contra la que se está evaluando porque obtenidosfrece el beneficio de que los parámetros del método de predicción no pueden ser ajustados exhaustivamente a casos particulares. Es por esto que se utiliza principalmente en situaciones de predicción, dado que intenta evitar que el aprendizaje se realice sobre un cuerpo de datos específico y busca obtener respuestas más generales.

La única desventaja que presenta es la necesidad esperable de correr los algoritmos en varias iteraciones, situación que puede tener un peso significativo si el método de predicción tiene un costo computacional muy alto durante el entrenamiento.

En primera instancia para probar el algoritmo de Knn utilizamos dos valores para K para analizar el comportamiento, esto lo veremos en mas detalle en la seccion de experimentacion.

2.3.3. ¿Qué pasaría si variamos la cantidad de particiones del k-fold?

Si particionamos en menos conjuntos, es decir, elegimos un K chico, lo que sucede es que vamos a tener particiones más grandes y eso resultará en que la base de entrenamiento será más pequeña. Luego el modelo no será tan robusto y se esperaría que las predicciones sean peores. Por ejemplo, si tengo una base de 100 imágenes y las divido en 5 particiones, cada partición va a tener 20 imágenes, van a ser utilizadas 20 imágenes para el testeo y 80 para el entrenamiento. En cambio si parto la base de datos en 2 particiones, van a quedar 50 imágenes para testear y 50 para el entrenamiento. Entonces, a mayor cantidad de imágenes para entrenar, mayor va a ser la probabilidad de tener una estimación de como verdaderamente funsiona el modelo.

Ese mismo razonamiento se puede utilizar a la inversa, si tengo mayor cantidad de particiones, voy a tener mayor cantidad de imágenes para entrenar y es muy probable que la los resultados obtenidos sean más fiables.

En otras palabras:

- A mayor cantidad de particiones, mayor cantidad de imágenes de entrenamiento y mayor fiabilidad.
- A menor cantidad de particiones, menor cantidad de imágenes de entrenamiento y menor fiabilidad.

Es importante también tener en cuenta, especialmente para aplicaciones más genéricas (por ejemplo detección de objetos de todo tipo en una imagen), tomar conjuntos heterogéneos. Si tenemos particiones con elementos similares entre si es posible que obtengamos mediciones de modelos muy buenos para detectar una característica pero muy malos para detectar otra. Supongamos que elegimos mal nuestras particiones y elegimos en cada partición todas imágenes correspondientes al mismo número. Entonces nuestras tazas de reconocimiento serían malas y el algoritmo poco efectivo. Esto también aplica a la inversa. Tomando buenas muestras, aunque sean pequeñas, podemos obtener muy buenos resultados si sabemos que elementos tomar y distribuirlos bien en todas las particiones.

2.3.4. Análisis de promedios

Para obtener un resultado global de los conjuntos de testing mencionados anteriormente, procedimos a realizar un promedio de los aciertos obtenidos para cada k y sacar el porcentaje de aciertos. A esto lo llamaremos, la tasa de efectividad. La ventaja de utilizar esta metrica, a diferencia de tomar los conjuntos por separado, es que al tener diversos conjuntos, se mitiga el problema de conjuntos para los cuales es muy efectivo la utilización de un k en particular, por las características de ese conjunto. Si el k resulta efectivo en el promedio de los conjuntos, nos indica que es efectivo globalmente y no para un caso en particular.

3. Experimentacion y resultados

Para analizar los algoritmos implementados vamos a utilizar varios archivos de test generados mediante Python como archivos de entrada, los cuales están más detallados en el apéndice, con el fin de realizar una serie de test que nos permitirán en primer caso encontrar parámetros buenos con lo que ejecutar los distintos métodos y posteriormente evaluar el desempeño de la implementación mediante las métricas propuestas por la cátedra.

Dividimos la experimentación en secciones, una para cada técnica. Y evaluaremos los parámetros individualmente para cada una de ellas

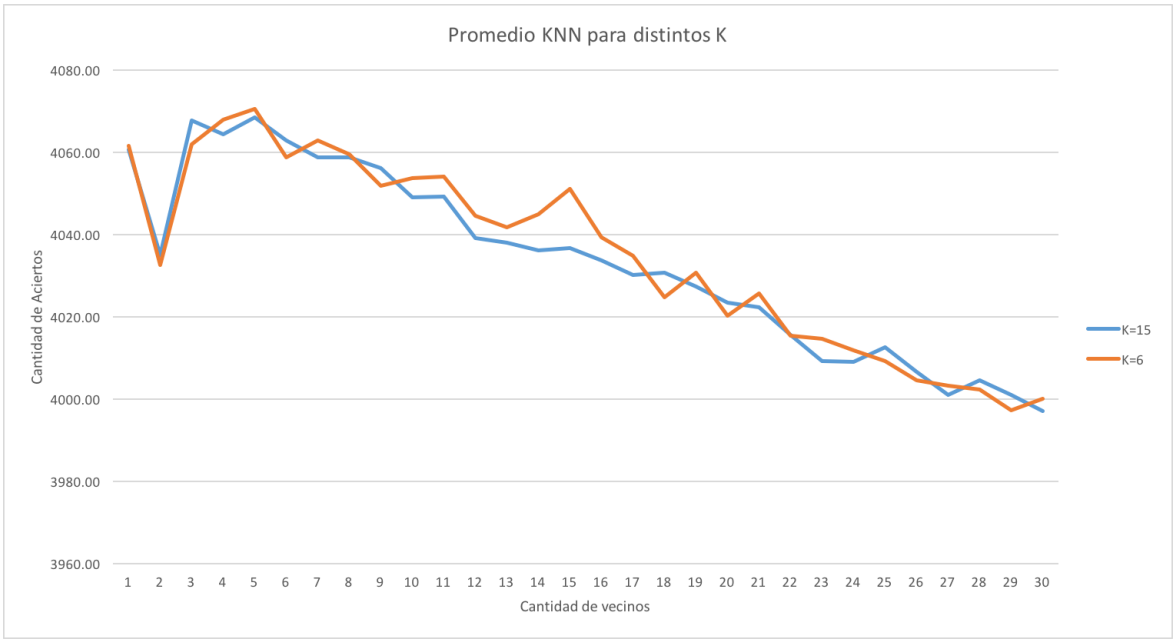
3.1. Algoritmo de K-NN

Lo primero que vamos a hacer es encontrar un valor de K y la cantidad de vecinos más cercanos k que nos permita maximizar la cantidad de aciertos, sin tener en consideración las métricas.

Ejecutamos el algoritmos de $K - NN$ variando los valores de k entre 1..30 dejando fija la cantidad de particiones para K =6 y K =15. Luego para cada una de las iteraciones tomamos el promedio. Esto se utilizó tanto para la cantidad de aciertos como para la cantidad de vecinos.

Cada uno de los conjuntos conto con 4200 imágenes a testear. En los siguientes gráficos presentamos algunos de los sets obtenidos:

Expresamos los aciertos para cada K en con el siguiente gráfico:

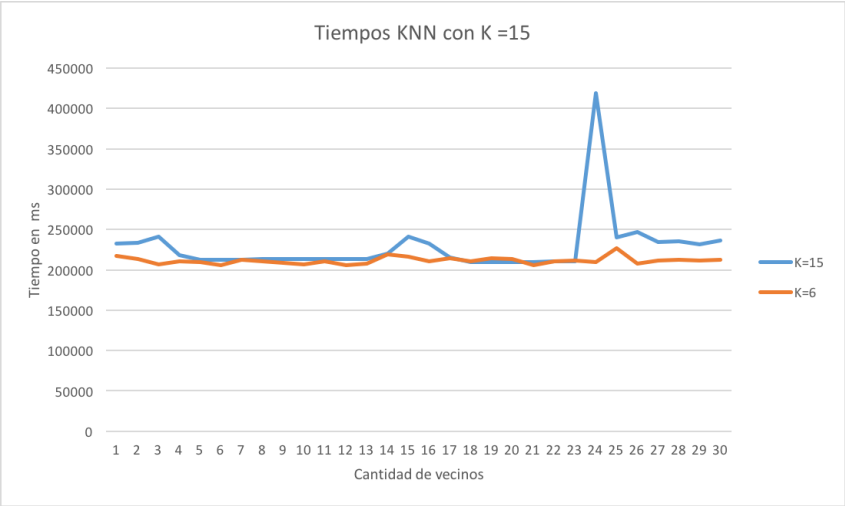


Como se puede observar para para ambos K se tiene el mismo patrón a lo largo que se incrementa la cantidad de vecinos k, pero ambas parecen alcanzar un máximo en $k = 5$ como cantidad de vecinos. Además notemos que a medida que se incrementa el valor de k la cantidad de aciertos va disminuyendo levemente, cumpliendo lo mencionado en el desarrollo. Cuanto más corta sea la distancia de los vecinos, más chances hay de tener un acierto sin importar el valor de K.

Si nos detenemos y observamos con más claridad cuando para la combinación de K=6 y k=5 es donde se maximizan en comparativa para knn, lo que intentaremos hacer en los próximos métodos es utilizar K=6 y ir variando la cantidad de vecinos más cercanos para ver si existe algún tipo de relación con el fin de asegurar que esa combinación es la mejor. Esto se debe a que no hay grandes diferencias

entre las dos particiones que utilizamos, muy probablemente si el K fuese mucho mayor la diferencia seria mayor.

Como segundo estudio, nos centramos en el análisis temporal para saber cómo afectaba la cantidad de particiones que tomamos para el Cross validation y variamos la cantidad de vecinos tomados para ver cómo se comportaba, los cuales arrojaron los siguientes resultados.

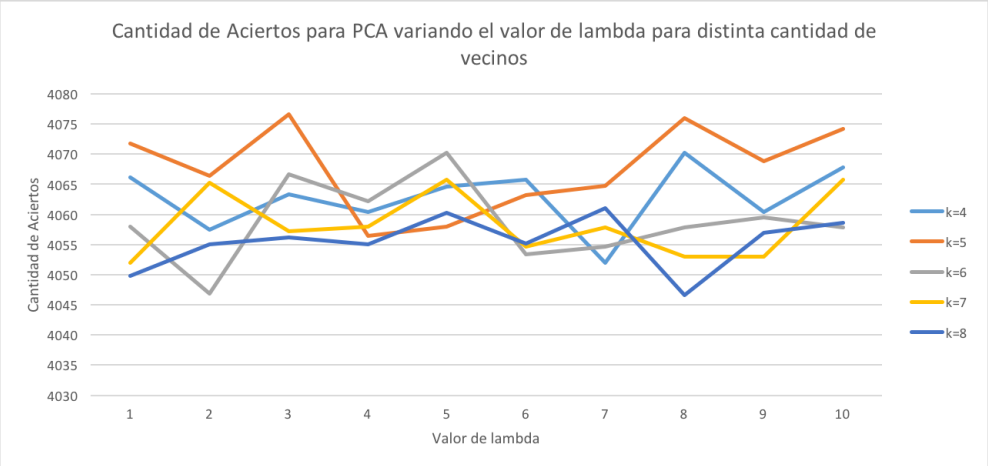


En cuanto a los tiempos para diferentes K no varían mucho en la media, si bien se puede notar al principio que a mayor cantidad de ejecuciones mayor es el tiempo que demora, algo previsible previamente.

Notemos que hay un pico, o crecimiento repentino, cuando tomamos entre k=23 y k=25 vecinos más cercanos. Sin embargo, esto es un caso que deberíamos seguir ejecutando para esos parámetros con el fin de ver si no es ruido que se pudo haber generado por otro proceso en el momento de la ejecución para descartar esto.

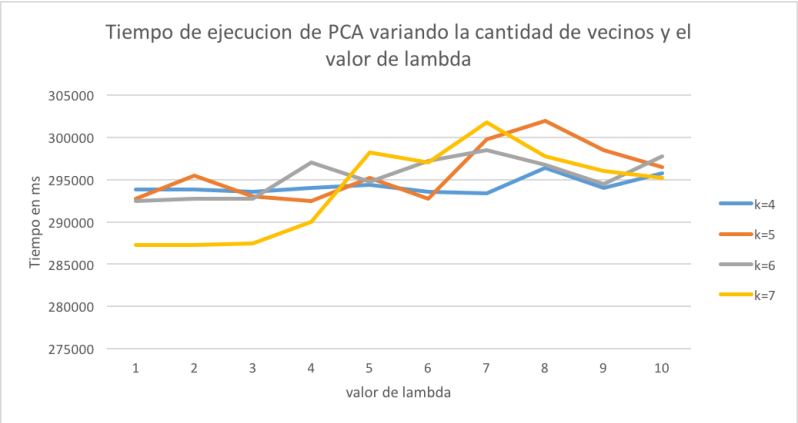
3.2. Algoritmo de K-NN con Optimización de PCA

Para el algoritmo de PCA lo que realizamos fue una variación de los valores de lambda (cantidad de componentes principales). Para esos valores de k medimos los tiempos de ejecución y los promediamos para poder ver de que manera varía la ejecución de los algoritmos en función de , luego tomamos un promedio de las ejecuciones y obtuvimos lo siguientes resultados:



Al ver estos resultados notamos a comparación de los resultados obtenidos para KNN con k=5 se respetan para la optimización, ya que mirando las combinaciones de lambda con cantidad de vecinos el que resulta como 'ganador' arrojando una cantidad de aciertos mayor a 4075 es cuando k=5 y λ=3.

A medida crecía el lamda no parece llegar a igualarlo pero si está muy cerca cuando se toman 8 componentes principales.



En cuanto a los tiempos cabe aclarar que estos no contemplan todo lo que se considera el 'entrenamiento' del sistema, es decir, todo el reprocesamiento que resultara en encontrar los valores principales. La justificación de esto es que el procedimiento se realizar a una vez, para entrenar el sistema y luego, al momento de clasificar las nuevas imágenes este tiempo podrá ser despreciado. Este grafico se puede ver que aumentar el produce un aumento lineal de los tiempos de ejecución, de lo que se desprende que aumentar la cantidad valores principales no resulta gratuito en términos de tiempo de ejecución y tiene cierto costo asociado.

De igual manera la distribución de tiempos al variar el lamda parece ir creciendo levemente y podríamos predecir que así va a ser su comportamiento, mientras más grande sea el valor de lambda mayor tiempo va a consumir.

3.3. Algoritmo de K-NN con Optimización de PSL-DA

Para el algoritmo de PLS-DA hemos continuado con el mismo estudio para poder realizar una comparativa entre ambos en cuanto a su taza de acierto. Si observamos los resultados de PLSDA pretendemos observar resultados similares a PCA en cuanto a cantidad de aciertos.

Luego de nuestros experimentos obtuvimos los siguientes resultados.

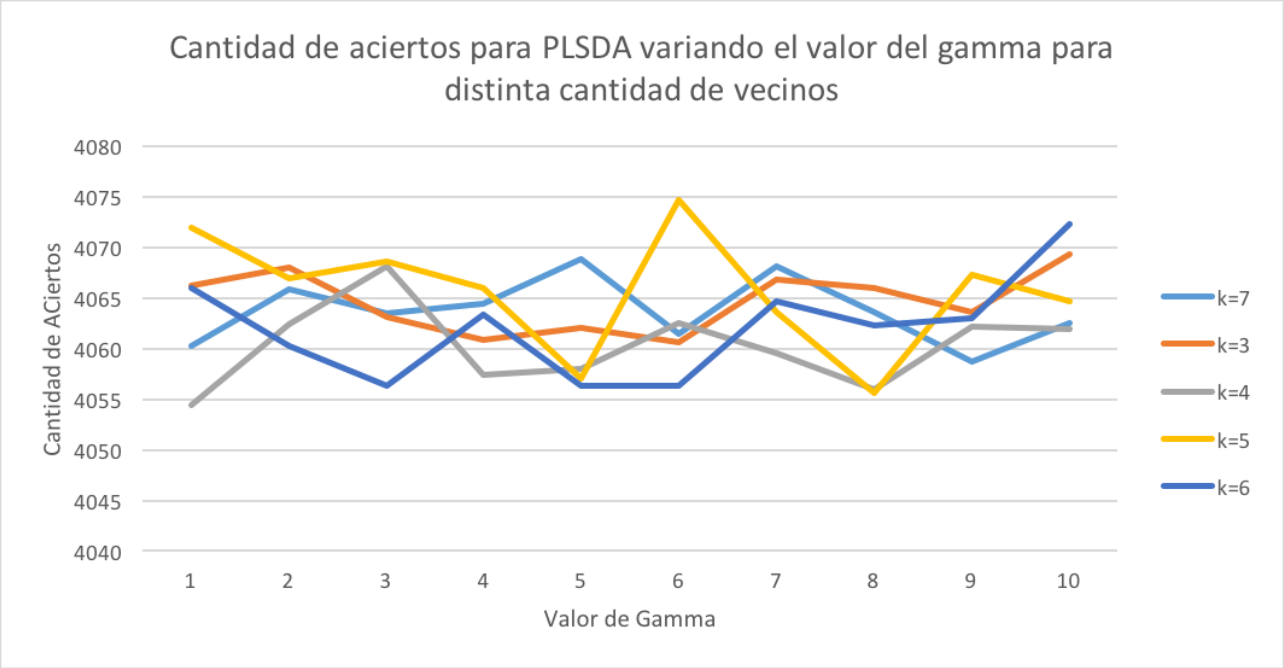
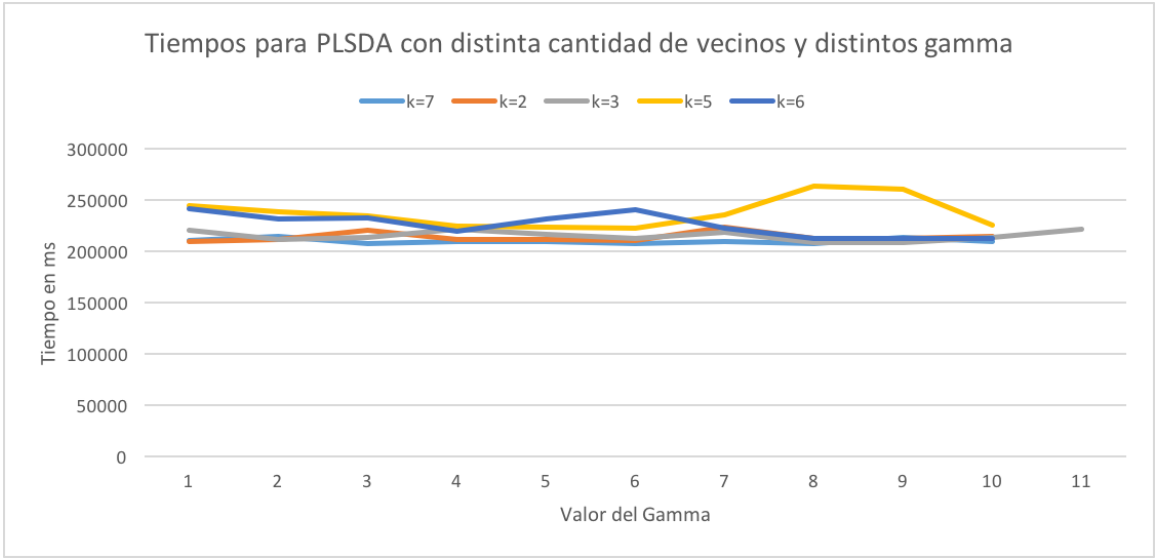


Figura 1: Comparacion de aciertos variando el gamma

De un modo muy parecido que para PCA con λ, podemos ver que si aumenta el γ, mejora la precisión pero si gamma aumenta demasiado , en algún momento empeora tu hit rate , suponemos que eso se debe a que si bien tenemos bastante información , la cantidad de vecinos no permite aprovecharla . Eso hace suponer que para que tengamos un buen hit rate , debe haber algún tipo de relación entre el k y el γ.

En cuanto a los mejores valores para la ejecución de PLSDA, sigue sucediendo que utilizando k=5 es cuando se maximizan la taza de acierto. En este caso es mas grande la cantidad de componentes que utilizamos a la de PCA y pero menor en cuanto a cantidad de aciertos. Esto puede deberse a que en la transformación estemos perdiendo calidad en los datos.



Por otro lado tenemos diferencias en cuanto a el tiempo de ejecución entre PCA y PLSDA, el segundo parece ser un poco mas estable a la variación de componentes. A su vez es menor en cuanto a la de

PCA, pero aquí es donde debemos analizar como podemos encontrar un balance en cuanto a tiempo de ejecución y cantidad de aciertos.

3.4. Precision, recall y F1 Score para los mejores resultados obtenidos

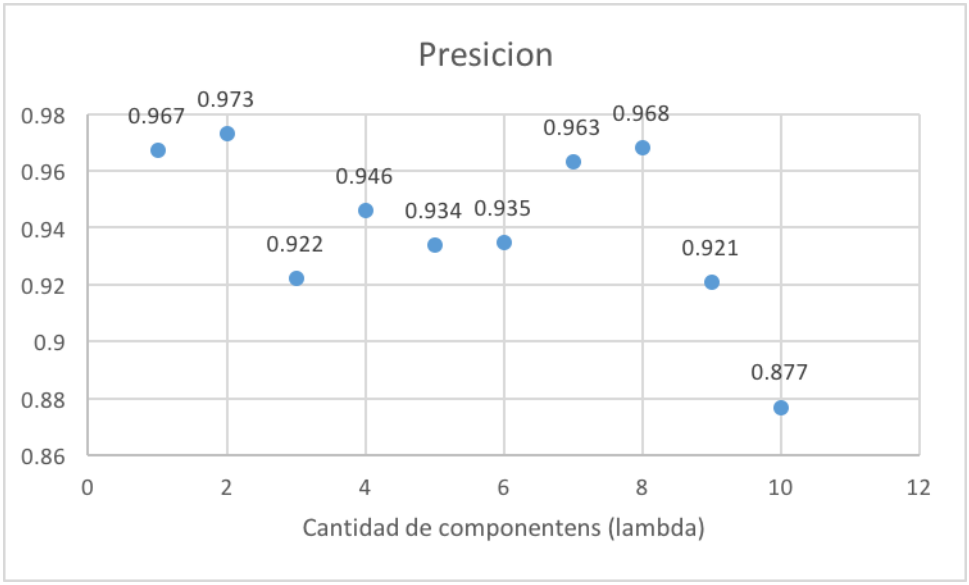
El ultimo de los puntos pedidos con el fin de analizar la calidad de los resultados obtenidos era la utilización de 2 métricas de las presentadas en el enunciado. Para nuestros experimentos decidimos usar Precisión y Recall.

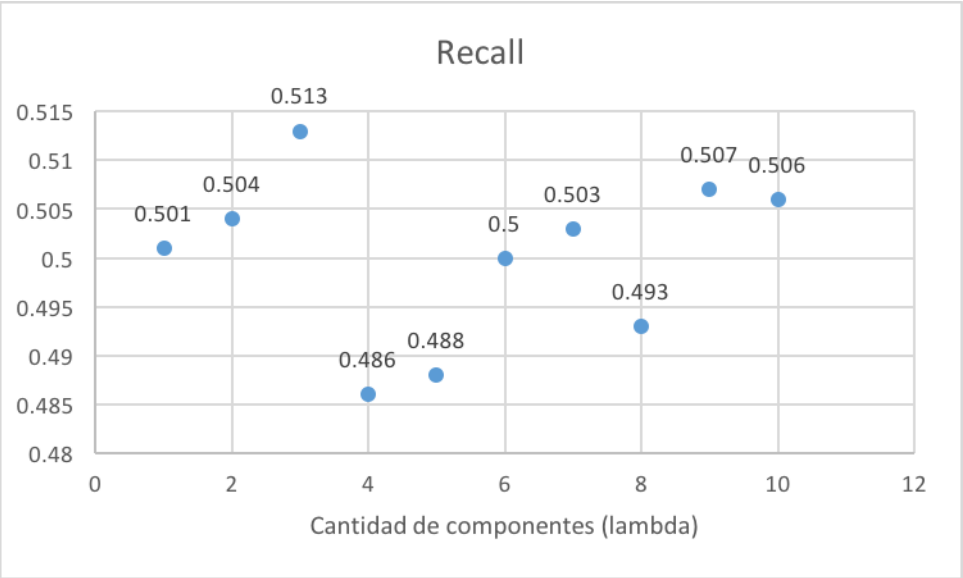
Precisión se basa en la cantidad de aciertos relativos dentro de una clase particular. Ósea, dada una clase i, definimos a los verdaderos positivos como tp_i y los falsos positivos como fp_i, estos son aquellos que fueron definidos como una clase a la cual no pertenecían. Con lo cual la precisión de dicha clase se define como

$\text{tp}_i/(tp_i + fp_i)$

Recall es una medida para saber qué tan bueno es un clasificador para identificar a los que pertenecen a una clase, asumiendo que fn_i son aquellos falsos negativos. El Recall de una clase está definido como $\text{tp}_i/(tp_i + fn_i)$

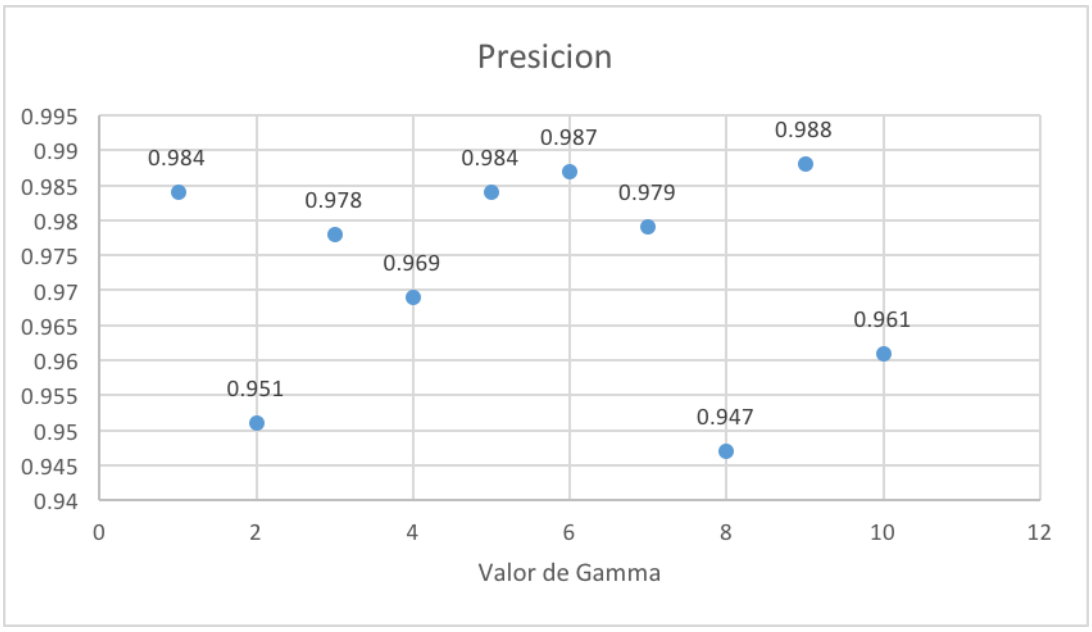
Luego para la mejor cantidad de vecinos obtenida (k=5); buscamos los valores de las métricas obtenidas para PCA.

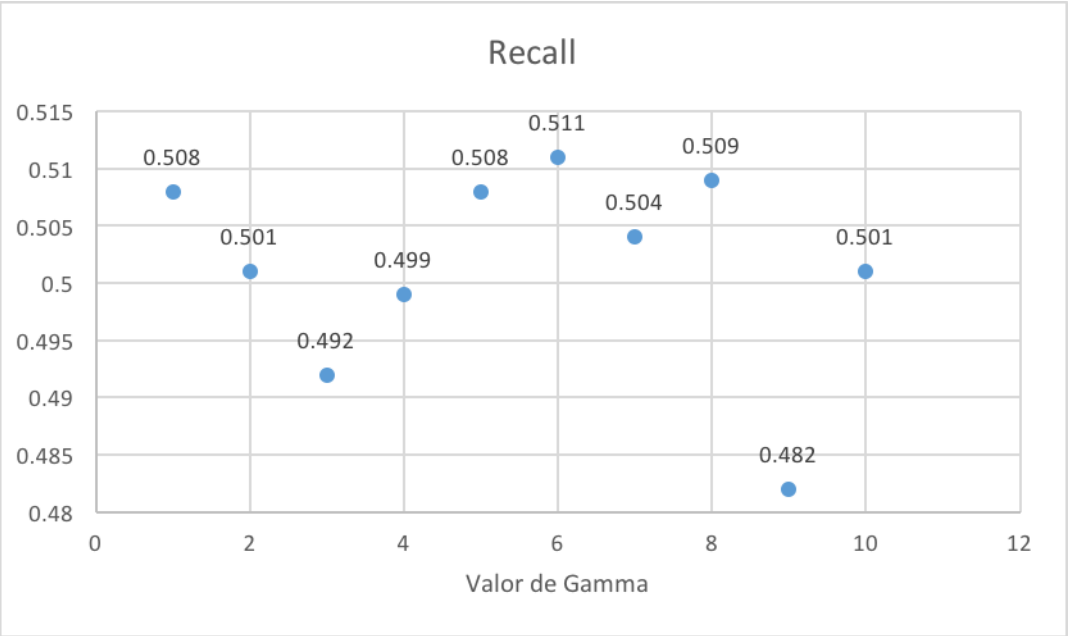




Como podemos ver los resultados para las métricas dieron muy parecidos en cuanto al lambda elegido para obtener el máximo dentro de la métrica. Sin embargo, esto está basado en que la cantidad de vecinos que tomamos es 5 dado a que en los experimentos de KNN+PCA demostrados con anterioridad fue con la cantidad de vecinos el cual obtuvimos los mejores resultados.

Por otro lado repetimos el análisis para PLSDA con el fin e analizar las mismas métricas





En este caso podemos sacar la misma conclusión que en el caso anterior en cuanto a que las métricas para la combinación de γ y k con la cual se maximiza es la misma que para la cantidad de aciertos.

Por lo que podemos definir como que las mejores combinaciones de parámetros para PCA y $PLS-DA$ son las dadas en este apartado.

4. Conclusiones

El análisis realizado nos lleva a sacar una serie de conclusiones en base a lo experimentado.

En primer lugar podemos mencionar que tanto para PCA y $PLSDA$ manejamos valores relativamente chicos, deberiamos de ver que sucederia si es que seguimos aumentando estos valores. De igual manera por mas de que los valores eran relativamente chicos los tiempos de ejecucion son previsibles debido a la cantidad de iteraciones que pueden realizar los algoritmos y a el tamaño de la entrada.

El algoritmo KNN presenta una gran efectividad, entendiendo que es una técnica que cuenta con varios años de antiguedad. Sin embargo, los tiempos necesarios para todas las comparaciones resultan considerablemente elevados. Como dato importante de destacar, entendemos que la efectividad de este algoritmo depende en gran medidad de la variación de los datos a analizar. En aquellos conjuntos donde la varianza es elevada y los datos se encuentran muy dispersos, promediar el resultado en base a sus vecinos más cercanos puede no resultar la mejor técnica a implementar. Lo mismo podria ocurrir en situaciones donde los datos se "asemejen" demasiado por la elección de las características a medir (situación que podría mitigarse eligiendo nuevas formas de representar los datos o realizando un preprocesamiento previo a estos).

Teniendo en cuenta esto, la relación costo-beneficio de la implementación y ejecución previa de una optimización como la de los algoritmos de PCA o de $PLSDA$, resulta mínima. Si bien es cierto que, como pudimos observar en el análisis, se pierde una efectividad, atribuimos este comportamiento a algunas de los supuestos que mencionamos que asumían los algoritmos.

Sin embargo, dada la característica principal de los algoritmos de PCA y $PLS - DA$ (ordenar las componentes principales en base a su relevancia), se permite ajustar la cantidad de datos a considerar, dando lugar a una mejora mas que considerable en la performance de aplicar sobre estos el algoritmo KNN y el uso de memoria. Como vimos durante nuestro análisis, la cantidad óptima está bastante por debajo del máximo y no tienen ningún beneficio considerar una mayor cantidad de estas.

Como resultado de esta característica, los tiempos de análisis se reducen drasticamente, todavía lejos de poder implementar este tipo de soluciones en "tiempo real" pero mucho más cercanos que utilizando solo el algoritmo de KNN.
Como se menciona al comienzo del trabajo y de este apartado, el preprocesamiento de las imágenes es otro factor que puede mejorar la eficiencia algorítmica. Así como PCA y $PLS - DA$ quitan ruido del dataset, es posible homogeneizar las imágenes por separado aplicando otros filtros.
Si bien el propósito del trabajo busca encontrar dígitos en imágenes este mecanismo se puede utilizar de un modo muy parecido para encontrar otras características tanto en imágenes como en audios y así etiquetar según clases que no tienen que ver necesariamente con la extracción de dígitos.

5. Apendice

Dentro de la carpeta experimentos encontraremos las siguientes carpetas que contienen datos de test los cuales se utilizaron durante la seccion experimentacion.

itemize

knn6 y knn15 son los que se utilizaron en la seccion de KNN para K=6 y K=15 respectivamente

pca y plsda contienen los datos de test para los distintos valores de k y γ para los test de estos metodos utilizados en las seccionesc correspondientes.

6. Bibliografía

- Numerical Analysis, Richard L. Burden & J. Douglas Faires, Chapter 6: Direct Methods for Solving Linear Systems.
- Machine Learning, Tom M. Mitchell.