DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

"CSI:DC"

Metodos numericos
Primer Cuatrimestre de 2016

Integrante	LU	Correo electrónico
Ricardo Colombo	156/08	ricardogcolombo@gmail.com
Diego Santos	874/03	diego.h.santos@gmail.com
Luis Badell	246/13	luisbadell@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. Introducción teórica

El objetivo de este trabajo práctico es desarollarar un clasificador que permita reconocer dígitos manuscritos. Par llevarlo a cabo analizaremos técnicas de reconocimiento óptico de caracteres (OCR). Es un proceso dirigido a la digitalización de caracteres manuscritos, los cuales identifican automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos.

Exploraremos la técnica de reconocimiento K-NN, K vecinos más próximos, además experimentaremos con distintas variantes de clasificación:

- PCA
- PLSDA

Luego analizaremos la performance de cada una de estas técnicas mediante un set de experimentos, para evaluar las fortalezas y deficiencias de cada una de ellas.

1.1. Introducción Teórica

1.1.1. Reconocimiento óptico de caracteres

La tecnología de reconocimiento de caracteres, OCR (Optical Character Recognition) engloba a un conjunto de téc nicas basadas en estadísticas, en las formas de los caracteres, transformadas y en comparaciones, que complementánd ose entre sí, se emplean para distinguir de forma automática entre los diferentes caracteres alfanuméricos existentes. Enrealidad no se reconocen exactamente los caracteres de un determinado alfabeto, sino que es posible distinguir entre cualquier conjunto de formas o símbolos. Sin embargo, se debe tener en cuenta que la precisión que se obtiene en la práctica al inten tar distinguir entre un conjunto de símbolos no es total. Por lo tanto, es fácil deducir que cuanto más numeroso es el conjunto de símbolos entre los que se debe decidir, mayor es la probabilidad de que se produzca un fallo de clasificación. En todo sistema de reconocimiento óptico de caracteres se distinguen al menos estas 4 etapas:

- Preproceso
- Segmentación
- Extracción de características
- Reconocimiento

Preproceso

En esta fase de preprocesamiento (o adecuación de la imagen) el objetivo que se persigue es eliminar de la imagen de cualquier tipo de ruido o imperfección que no pertenezca al carácter, así como normalizar el tamaño del mismo. La normalización de la imagen también puede implicar un binarizado de la misma. Para la eliminación del ruido que puede aparecer en una imagen digital se utilizan diversos algoritmos:

- Etiquetado: para la división de la imagen en regiones de componentes conectadas.
- Erosión / expansión: para la eliminación de pequeños grupos de píxeles.

- Umbralizado de histograma: para eliminar/seleccionar los objetos más brillantes o más oscuros de la imagen.

Segmentación

Una vez preprocesada la imagen se deberá segmentar en las diferentes componentes conexas (parte de la imagen donde todos los píxeles son adyacentes entre sí) que la componen. La segmentación de la imagen constituye una de las mayores dificultades del reconocimiento, y se hace necesaria para poder reconocer cada uno de los caracteres de la imagen binaria. La fragmentación o segmentación es la operación que permite la descomposición de un texto en diferentes entidades lógicas. Estas entidades deben ser lo suficientemente invariables, para ser independientes del escritor, y lo suficientemente significativas para su reconocimiento.

Extracción de características

Una vez realizada la segmentación, se tiene una imagen normalizada en la que se encuentra la información susceptible de ser "reconocida". La información así representada, una matriz bidimensional de valores binarios.

La extracción de las características es una de las fases más difíciles, dado que es muy difícil elegir un conjunto de características óptimo. Para que una característica se pueda considerar buena debe tener:

- Discriminación: Deben ser características que diferencien suficientemente una clase de otra
- Deben tener igual valor para mismas clases
- Independencia: Las características deben estar incorreladas unas de otras.
- Pequeño espacio para características: El número de características debe ser pequeño para la rapidez y facilidad de clasificación.

En este trabajo desarrollaremos y evaluaremos dos técnicas.

`PCA (Principal Component Analysis)`

El objetivo de esta técnica es definir una transformación lineal desde el espacio de representación original a un nuevo espacio en el que las distintas clases de las muestras quedan mejor separadas. Esta transformación permite reducir la dimensión del nuevo espacio sin perjudicar sensiblemente la capacidad discriminativa de la nueva representación.

`PSL-DA (Partial Least Squares Discriminant Analysis)`

Es un método estadístico que tiene relación con la regresión de componentes principales, se encuentra una regresión lineal mediante la proyección de las variables de predicción y las variables observables a un nuevo espacio. Debido a que tanto los datos de X e Y se proyectan a nuevos espacios, los familia de los modelos PLS se conoce como factor de modelos bilineales. Los cuadrados mínimos parciales Análisis discriminante (PLS-DA) es una variante que se utiliza cuando la Y es binaria.

Reconocimiento

Una vez se tienen las características más importantes de la imagen a analizar hay que determinar el carácter correspondiente en este trabajo exploraremos la técnica de KNN.

1.1.2. K-NN (K vecinos más próximos)

Es un método no paramétrico y supervisado, que dado un conjunto de objetos prototipo de los que ya se conoce su clase (es decir, dado un conjunto de caracteres de muestra) y dado un nuevo objeto cuya clase no conocemos (imagen de un carácter a reconocer) se busca entre el conjunto de prototipos

los “k” más parecidos a nuevo objeto. A este se le asigna la clase más numerosa entre los “k” objetos prototipo seleccionados.

Entrenamiento y Test

Conociendo el funcionamiento básico del método de clasificación de los “k vecinos más próximos” es obvio que para poder empezar a trabajar con este método es necesario reunir un conjunto de datos etiquetados, es decir, un conjunto de muestras prototipo con las clases a las que pertenecen. Esta recolección implica disponer de una base de datos de imágenes de los tipos de caracteres que posteriormente se esperen reconocer. A este conjunto de datos se le denomina conjunto de entrenamiento. Sin embargo, la fase de entrenamiento no solo consiste en la recopilación de estos datos, sino que, típicamente, los datos originales que se dedican al entrenamiento deben ser preprocesados adecuadamente para obtener representaciones compactas y coherentes. Esto quiere decir que las imágenes deben ser segmentadas, normalizadas y transformadas para obtener los vectores de baja dimensionalidad que finalmente se almacenan como conjunto de entrenamiento. Con este conjunto de entrenamiento ya construido, el clasificador “knn” ya puede ser utilizado para reconocer la clase de una nueva muestra. Esta es la fase de test y lógicamente, también aquí es necesario aplicar todo el preproceso descrito anteriormente a cada una de las nuevas muestras. Por lo tanto, aquí se ve la necesidad de disponer de métodos rápidos de realizar estas tareas de preproceso, puesto que la velocidad de reconocimiento dependerá, en parte, de ellos. En la práctica se tiene que este preproceso es posible realizarlo muy rápidamente, aunque justo a continuación aparece la parte del proceso de reconocimiento que normalmente más carga computacional conlleva, la clasificación.

2. Desarrollo

Para cumplir con el objetivo del trabajo práctico, lo primero que desarrollamos fue la implementación del algoritmo de $K - NN$. Como mencionamos en la introducción esta tecnica permite dado un dado del que no conocemos a que clase pertenece, buscar entre el set de datos de imagenes etiquetadas las k más parecidas, denomidadas *vecinas*. Luego una vez identificados determinar cual es la moda.

2.1. Elección del K

La mejor elección de k depende fundamentalmente de los datos; generalmente, valores grandes de k reducen el efecto de ruido en la clasificación, pero crean límites entre clases parecidas. Un buen k puede ser seleccionado mediante una optimización de uso.

La exactitud de este algoritmo puede ser severamente degradada por la presencia de ruido o características irrelevantes, o si las escalas de características no son consistentes con lo que uno considera importante.

2.2. Algorimo

TP1 1 vector KNN (matriz etiquedados, matriz sinEtiquedar, int cantidadVecinos)

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

Para encontrar las etiquetas implementamos el siguiente algoritmo

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

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

2.3. Optimización con PCA

El Análisis de Componentes Principales 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 proyecci on de los datos queda ubicada como la primer coordenada, la segunda mayor varianza en la segunda posición, y asi sucesivamente.

En este sentido, PCA calcula la base más significativa para expresar nuestros datos. De esta manera entonces, será fácil quedarnos con los 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 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 $M_{i,j} se obtenga el valor de la covarianza del pixel i contra el pixel j$.

Luego, utilizando el método de la potencia, procederemos a calcular los primeros autovectores de esta matriz. Una vez obtenidos los autovectores, multiplicando cada elemento por los 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. Esto se suma a mejores tiempos de ejecuci on, 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: asume que estos estadísticos describen la distribución de los datos sobre el eje.
- Varianza alta tiene una dinámica importante: Varianza alta significa senal. 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.

2.3.1. Cross-Validation

Para medir la precisión de los 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 est a evaluando porque ofrece

el beneficio de que los parámetros del método de predicción no pueden ser ajustados exhaustivamentea 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.

2.3.2. Algoritmos para Optimización con PCA

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

```
1: matriz covarianza = obtenerCovarianza(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
10:
11: return maximo(numeros)
```

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

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
```

2.4. Optimización con PLS-DA

2.4.1. Algoritmos para Optimización con PLS-DA

3. Experimentación

Para analizar los algortimos implementados vamos a utilizar los archivos test1.in y test2.in provistos por la cátedra y realizar una serie de test que nos permitirán en primer caso encontrar parametros buenos con lo que ejecutarlos y posteriormente evaluar el desempeño de la implementación mediante las métricas propuestas por la cátedra.

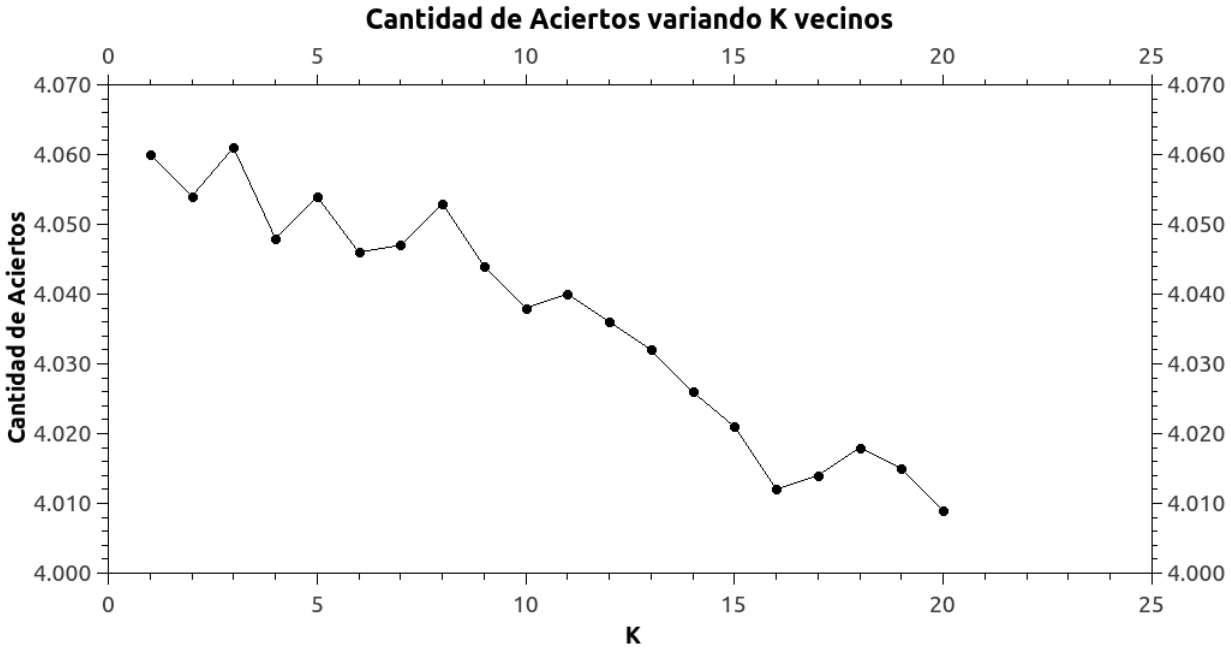
Dividimos la experimentación en dos secciones, una para cada archivo de prueba. Y evaluaremos los parametros individualmente para cada una de ellas

3.1. Algoritmo de K-NN

Lo primero que vamos a hacer es encontrar un valor de 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..20 dejando fija la cantidad de particiones en 10. Luego para cada K nos quedamos con los resultados de la iteración con mas aciertos.

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



Como se puede observar la iteración que mas aciertos dió es para $K = 3$. Además se puede observar que a medida que se incrementa el valor de K la cantidad de aciertos va disminuyendo levemente, cumpliendo lo mencionado en la introducción teórica. Cuanto mas corta sea la distancia de los vecinos, mas chances hay de tener un acierto.

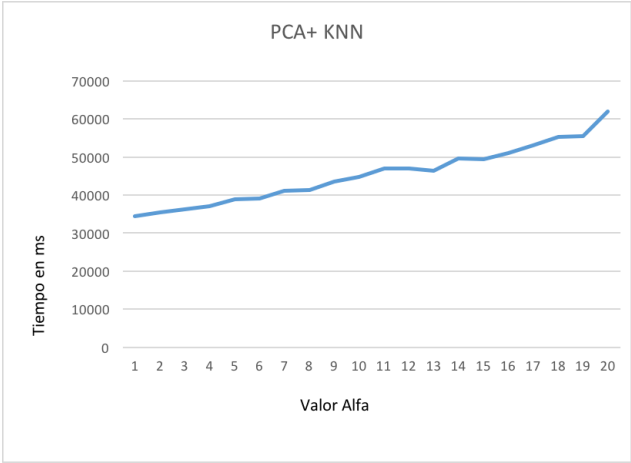
Luego de los valores obtenidos en base a estas pruebas tomamos las siguientes metricas solicitadas por la catedra.

Prescion: Es una medida de cuantos aciertos relativos tiene un clasificador dentro de una clase particular.

Recall: Es una medida de que tan bueno es un clasificador para, dada una clase particular, identificar correctamente a los pertenecientes a esa clase.

3.2. Algoritmo de K-NN con Optimización de PCA

Para el algoritmo de PCA lo que realizamos fue una variacion de los valores de alfa en el algoritmo de pca. para eso $k - fold$ medimos los tiempos de ejecución y los promediamos para poder ver de que manera varía la ejecución de los algoritmos en función de α, luego tomamos un promedio de las ejecuciones y obtuvimos lo siguientes resultados:



Cabe aclarar que estos tiempos no contemplan todo lo que se considera el 'entrenamiento' del sistema, es decir, todo el preprocesamiento que resultara en encontrar los valores principales. La justificacion de esto es que el procedimiento se realizar a una vez, para entrenar el sistema y luego, al momento de clasificar las nuevas imagenes este tiempo podra ser despreciado.Este grafico se puede ver que aumentar el produce un aumento lineal de los tiempos de ejecucion, de lo que se desprende que aumentar la cantidad valores principales no resulta gratuito en t erminos de tiempo de ejecuci on y tiene cierto costo asociado.

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

Habiendo fijado k = 3 , corremos el test1.in variando el gamma utilizando los valores 1,2,10 y 50. Aqui el grafico

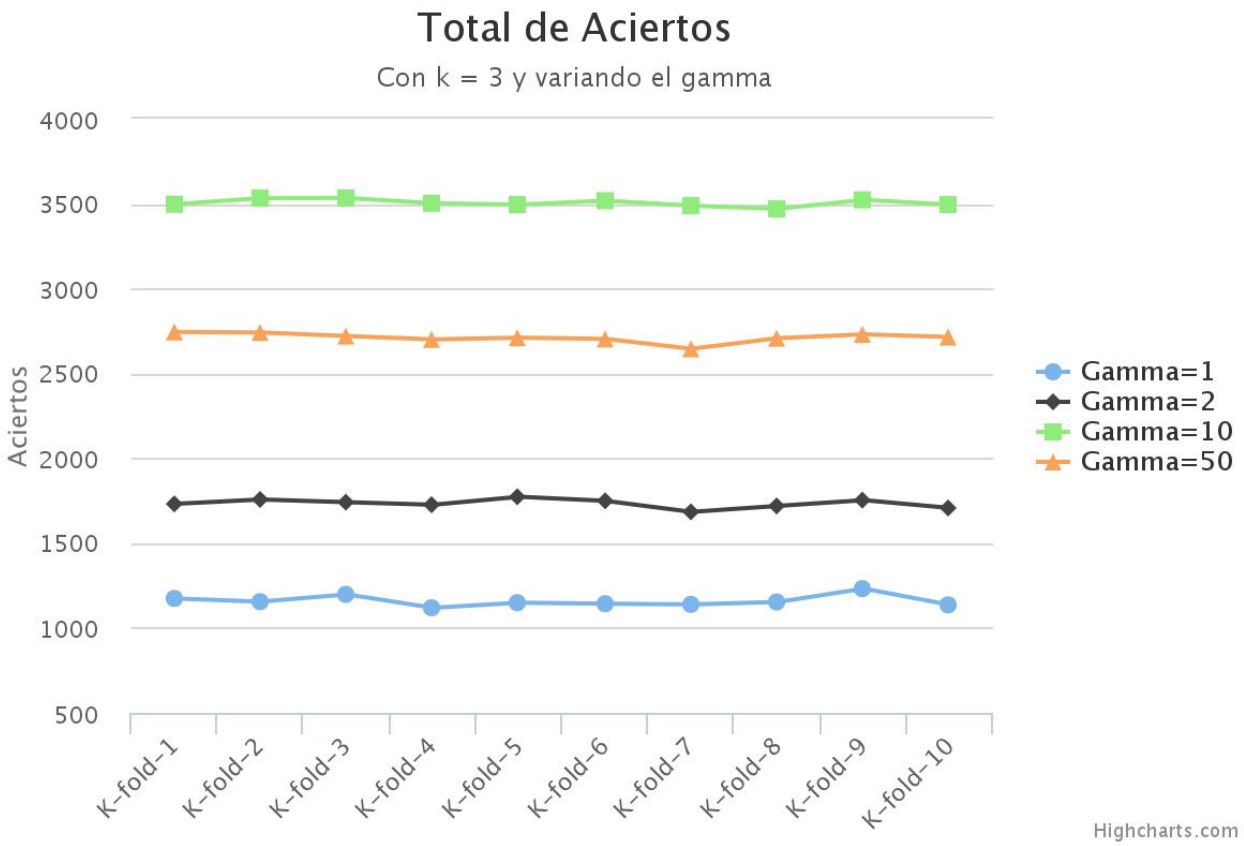


Figura 1: Comparacion de aciertos variando el gamma

Vemos que si aumenta el gamma , mejora la precision pero si gamma aumenta demasiado , en algun momento empeora tu hit rate , suponemos que eso se debe a que si bien tenemos bastante informacion , la cantidad de vecinos no permite aprovecharla . Eso hace suponer que para que tengamos un buen hit rate , debe haber algun tipo de relacion entre el k y el gamma . Eso se va a poner a prueba usando el test2.in

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

El experimento que nos planteamos es el siguiente. Dejamos fijo el gamma en 13 y hacemos variar el k . Aqui el grafico

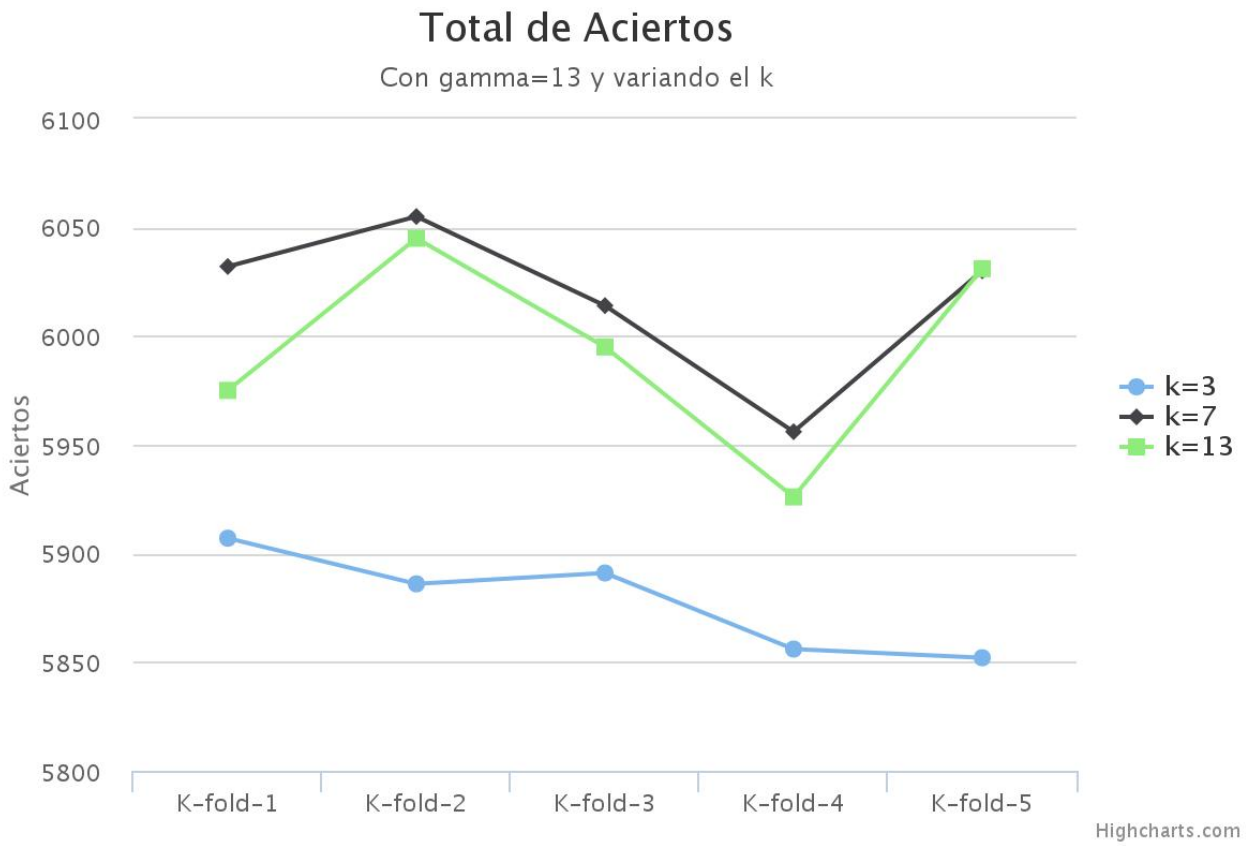


Figura 2: Comparacion de aciertos variando el k

De 3 a 7 mejora el hit rate pero es muy poca la mejora en comparacion al aumento de tiempo de computo