



**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 learningn 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.

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

Índice

1. Introduccion Teorica	3
2. Desarrollo	4
2.1. K vecinos mas cercanos	4
2.1.1. Identificacion de vecinos	4
2.2. Optimización mediante el analisis de componentes principales	5
2.2.1. Algoritmos para Optimización con PCA	6
2.3. Optimización mediante la regresion por Cuadrados Minimos Parciales	8
2.3.1. Algoritmos para Optimización con PLS-DA	8
2.3.2. Cross-Validation	8
2.3.3. ¿Qué pasaría si variamos la cantidad de particiones del k-fold?	9
2.3.4. Análisis de promedios	9
3. Experimentacion y resultados	10
3.1. Algoritmo de K-NN	10
3.2. Algoritmo de K-NN con Optimización de PCA	10
3.3. Algoritmo de K-NN con Optimización de PSL-DA	11
3.4. Algoritmo de K-NN con Optimización de PSL-DA	12

1. Introduccion Teorica

Dada las ultimas tecnologías presentadas en el mundo de las redes sociales y ambiente de sistemas en general, se ha incrementado el auge de aquellas que tienen el manejo de imágenes de por medio, dentro de las más conocidas (Facebook) se utiliza el reconocimiento de caras dentro de una imagen para poner etiquetas a aquellas caras ya conocidas o aquellos sistemas de validación como los conocidos CAPTCHA para el reconocimiento de textos manuscritos. Sin ir más lejos, estas grandes compañías utilizan distintos métodos numéricos para poder implementar este tipo de clasificaciones a la hora de poner etiquetas a imágenes ya conocidas, y sin que los usuarios se dieran cuenta estuvieron entrenando estos algoritmos con el fin de mejorar su tasa de acierto.

En el contexto de la materia de métodos numéricos intentaremos ir con un enfoque no muy diferente pero un dominio de cantidad de etiquetas y tamaño de imágenes un poco más reducido ,pero no muy alejados de la realidad, utilizando este tipo de técnicas de machine learning con el fin de poder dar con la clasificación de dígitos manuscritos. Dada una base de datos provista por la catedra con una gran cantidad de imágenes etiquetadas, intentaremos dar con estas utilizando dichas técnicas explicadas en los siguientes párrafos y detalladas con mayor profundidad en las sección de desarrollo.

En primer lugar Exploraremos la técnica de clasificación por **K vecinos más cercanos** (kNN dado las sigla en ingles). Este método a grandes rasgos se basa en dado el dominio de datos conocidos (etiquetados), se intenta de dar con aquellos que no tengan etiqueta basándonos en los que estén más próximos a él en el espacio vectorial mediante la función de distancia, donde las dimensiones del espacio están atadas al tamaño de la imagen, en nuestro caso R^{784} dado a que las imágenes son de 28×28 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. A diferencia de estas grandes compañías, nuestro conjunto de datos es acotado, reduciendo así los tiempos de computo pero agregando una complejidad en cuanto a que imágenes vamos a reconocer.

Para esto utilizaremos un método denominado Cross validation. Este procedimiento se basa en que, dado el conjunto de imágenes realizaremos una partición con el fin de entrenar nuestro algoritmo. Esta división del conjunto se utilizara como entrenamiento, una parte, y el restante como test 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 el 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 eucladiana. Sea un instancia representado como un vector $x = (x_1, x_2, \dots, x_n)$, 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 "parecidos."^a nuestra instancia.

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

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

donde $\delta(a,b)$ devuelve 1 si $a=b$ y 0 caso contrario.

2.1.1. Identificacion de vecinos

Para este trabajo en particular tomamos 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). definimos que dos imágenes son "parecidas" 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. La etiqueta para esta imagen vendrá dada por la moda.

Para los siguientes pseudocódigos será necesario asumir que todas las estructuras utilizadas almacenan datos enteros 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 sinEtiquetar,int cantidadVecinos)

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

A priori podemos suponer que al ir incrementando la cantidad de vecinos (o sea, 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).

Sin embargo, y mas allá de las mejoras que puedan realizarse sobre los datos en crudo, este algoritmo

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(etiquetadosi,incognita)
4:   colaPrioridad.push((norma(resParcial),etiqueta(etiquetadosi))
5: end for
6: vector numeros = vector(10)
7: while cantidadVecinos>0 & noesVacía(resultados) do
8:   int elemento =primero(resultados.etiqueta)
9:   numeroselemento ++
10: end while
11: return maximo(numeros)
```

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)
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:     nuevoVectorj =  $entrada_{(j,i)}medias_i$ 
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 += entradai,j
5:   end for
6: mediasi = suma/cantFilas(entrada)
7: end for
8:
9: return medias
```

2.3. Optimización mediante la regresión por Cuadrados Mínimos Parciales

Por último analizaremos la otra transformación, que si bien es similar al anterior (**PCA**), la principal distinción entre ambos es que este utiliza las clases para influenciar el traspaso al nuevo espacio de variables, convirtiéndolo así en un método supervisado. El método PLS-DA consiste en una regresión clásica PLS, que permite el manejo de datos multicategoricos a diferencia de PLS que es bilineal.

Si estuviéramos definiendo el PLS clásico deberíamos definir X de igual manera que se hizo en el punto anterior, o sea 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 covarianza entre las muestras y las clases en el nuevo espacio quedándonos de esta manera.

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

Si definimos como t y u los vectores de las matrices de transformación 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 matriz $X^t Y Y^t X$.

Una vez realizado esto utilizaremos en método de deflación para quitar este autovector para así obtener el primero de los γ mayores autovectores mediante un esquema iterativo.

2.3.1. Algoritmos para Optimización con PLS-DA

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 obtenidos se ve 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 hicimos es decir, siempre experimentamos en 10 particiones distintas y nunca variamos el K . Tomamos la primera partición de las 10 y utilizamos las otras 9 para entrenar, luego tomamos la segunda partición y utilizamos la primera partición y las otras 8 particiones para entrenar, y así sucesivamente, hasta poder testear con todas las distintas particiones.

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 funciona 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 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 tasas 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 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.

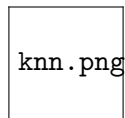
Dividimos la experimentación en dos secciones, una para cada archivo de prueba. Y evaluaremos los parametros individualmente para cada una de ellas

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.

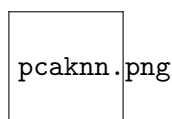
Luego de los valores obtenidos en base a estas pruebas tomamos las siguientes metricas solicitadas por la catedra.

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 realiza a una vez, para entrenar el sistema y luego, al momento de clasificar las nuevas imágenes este tiempo podrá ser despreciado. Este gráfico se puede ver que aumentar el γ produce un aumento lineal de los tiempos de ejecución, de lo que se desprende que aumentar la cantidad de valores principales no resulta gratuito en términos de tiempo de ejecución 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 γ utilizando los valores 1, 2, 10 y 50. Aquí el gráfico

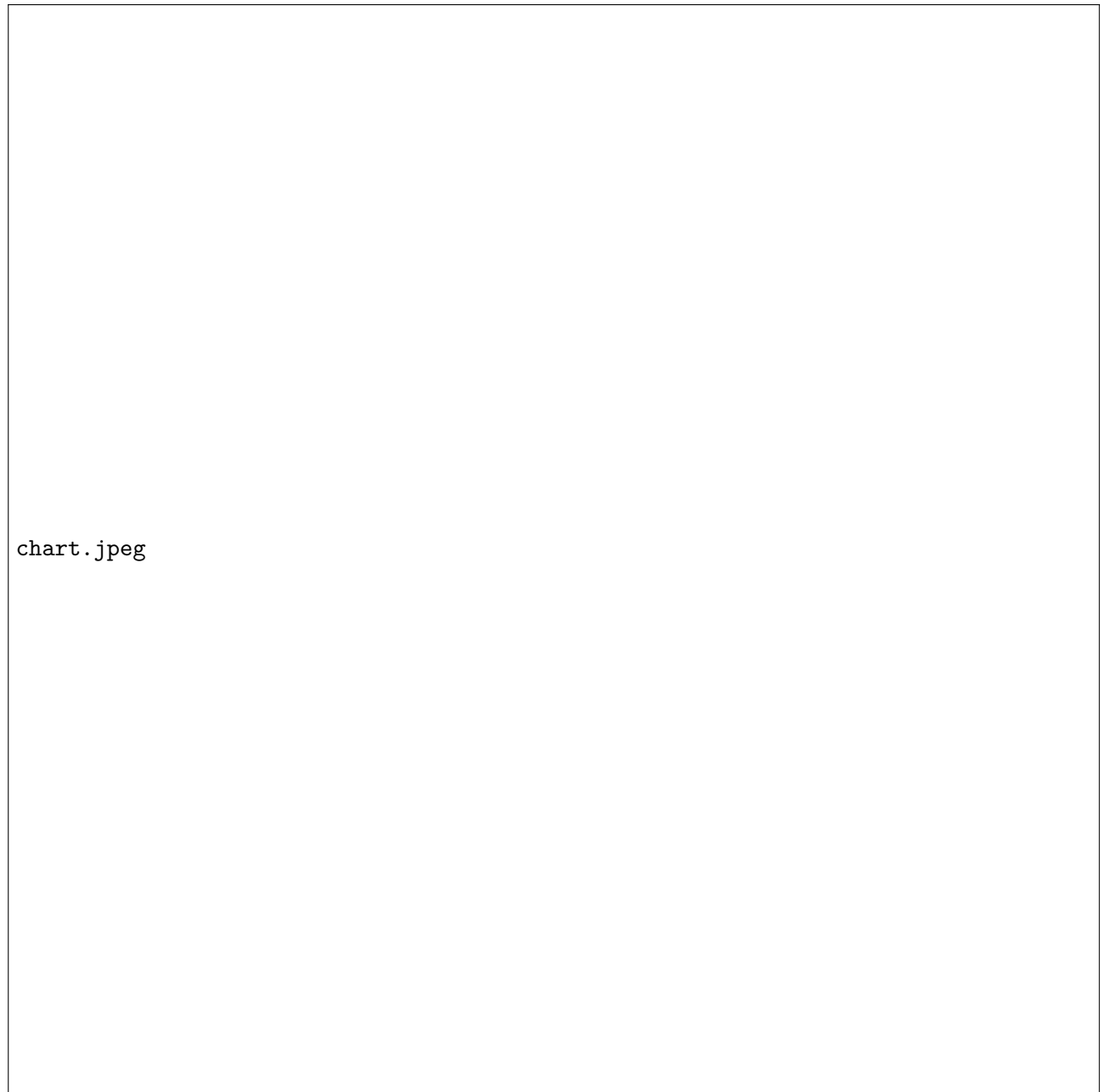



Figura 1: Comparación de aciertos variando el γ

Vemos que si aumenta el γ , mejora la precisión pero si γ aumenta demasiado, en algún

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



chart(2).jpeg

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