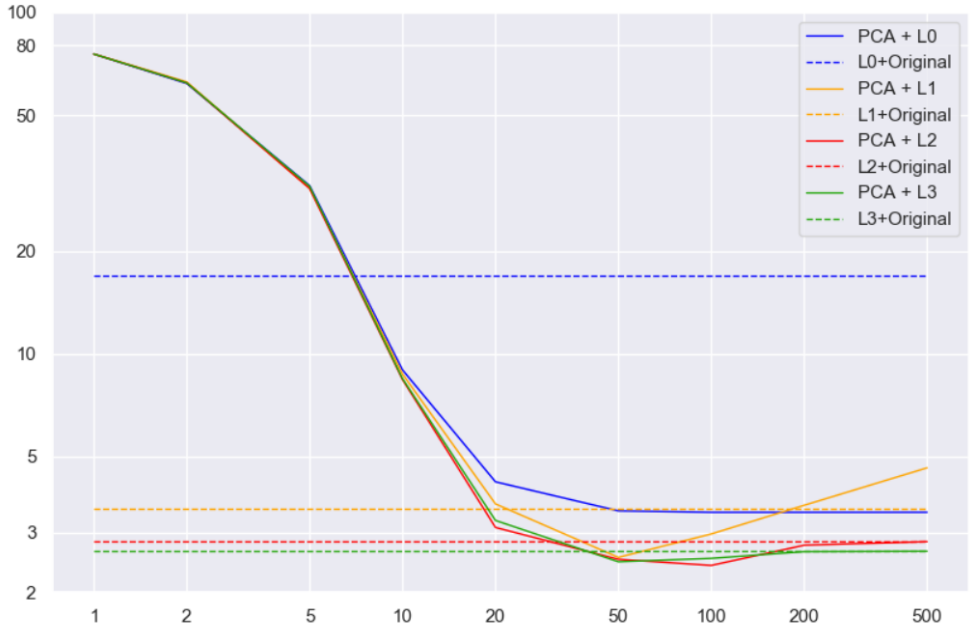
Ejercicios opcionales en orden [1,3,2]: Se nos plantean varios ejercicios opcionales donde hay que implementar nuevas distancias así como nuevas técnicas de optimización de error.

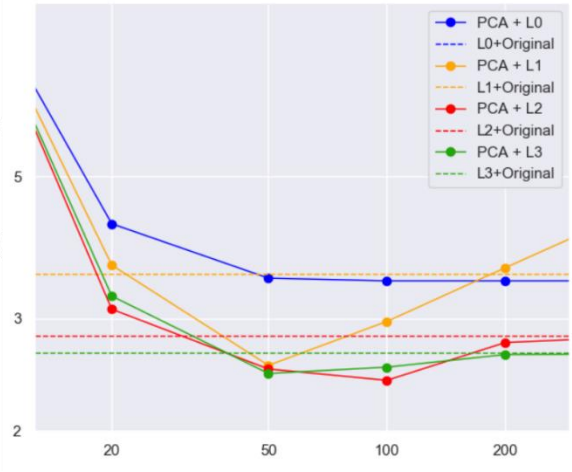
Siguiendo el orden de los ejercicios opcionales, nos encontramos primero con las distancias, las cuales son distancia L0dist.m , L1dist.m y L3dist.m correspondientes a distancias 0, 1 y 3 respectivamente. Con estas implementaciones trataremos de optimizar el K-NN que tenemos, por ende, todo el código así como las pruebas se quedarán igual, solo cambiaremos la función de distancias que se llamará desde el knn+pca-exp.m. Por lo tanto, nuestro experimento lo hemos separado en dos partes, primero calculamos el error producido al usar cada distancia sin usar el PCA y luego la progresión de error usando el PCA con los distintos k del rango [1, 2, 5, 10, 20, 50, 100, 200, 500]. Después de calcular los resultados para cada distancia, las hemos representado en la siguiente gráfica para ver las relaciones existentes con la distancia L2dist.m así como entre ellas mismas.



Lo primero que se observamos es la distancia L0, que como vemos esta se mantiene constante a lo largo de todas las iteraciones superiores a k = 50 con incremento de k. Tampoco converge con su versión original, lo que nos muestra que el PCA mejora bastante la distancia L0, ya que por la naturaleza del cálculo de la L0, la reducción de dimensionalidad favorece que los resultados mejoren. Pero como observamos, tiene casi 1% más de error que las otras distancias, por lo tanto, la aplicación de L0 con PCA podría ser una opción, pero en esta comparativa la descartaremos por ahora, ya que incluso añadiendo un margen de error para el uso de la distancia L0 en datos de producción, suponiendo que clasificaremos cosas parecidas al MNIST, el error no debería de irse tanto tampoco.

Ahora bien, las distancias interesantes son las de L1 y L3 con PCA que muestran resultados competitivos frente a la distancia L2 con PCA. Si nos acercamos más al punto donde se empiezan a intercedir (ver tabla de abajo), podemos observar que dependiendo del número de K que tengamos, una u otra implementación es mejor. Así pues, obsérvese que tanto la distancia L2 como la L3 van a converger al final a su respectiva distancia original, cosa que no pasa con la distancia L1, además si nos fijamos en la L1, veremos que no tiende a normalizarse su error, es decir, una vez alcanzado su mínimo en K=50 para

nuestro ejemplo, empieza a crecer el error drásticamente de una forma lineal a lo largo de los K. Con esto podríamos decir que, por una parte tenemos que usar con cuidado el PCA con L1, ya que dependiendo de la cantidad de vecinos que cojamos nos puede beneficiar el error o, sobrepasado el límite nos empeorará mucho el error, incluso tendremos peor modelo usando PCA que sin usando la implementación original de L1, ya que no converge con su implementación original. Por otra parte, la distancia L1 nunca llega a mejorar el error obtenido con las distancias L2 y L3 con PCA, llevando a pensar que su implementación con PCA no sería rentable, pero debemos fijarnos en K=50, donde la L1 se diferencia por muy poco de las demás distancias, así pues podría darse el caso que para datos concretos nos de menor error que las otras dos distancias. Además, para K=50, la L1 muestra buenos resultados frente a las implementaciones originales sin PCA de L2 y L3, con lo que si lo que buscamos es menor carga de cálculo tolerando algo de error, la L1 en su versión con PCA nos podría ser muy útil, al obtener un resultado similar con menos iteraciones que sus comparativas L2 y L3 con PCA. Comparando ahora la L2 con la L3, podemos observar que las dos convergen a su implemetnación original como dijimos antes, pero tambien se ve que la L3 converge mucho antes que la L2, por ello tiene una curva menos concava, lo que nos lleva a pensar que la L2 es mejor en la comparación final, es decir, si nos fijamos en la grafica de arriba, a la vez que en la tabla mostrada a la izquierda, podemos observar que la diferencia entre L2 y L3 es de 0.06%, lo que quiere decir que escoger una u otra no tiene mucha diferencia a error se refiera. Por otra parte, si suponemos que las ejecuciones se realizan en lenguajes rapidos como el C o C++, y el código está perfectamente optimizado, pues la diferencia entre calcular los K=50 frente a calcular los K=100 es mínima, por ello, referiendonos al caso de MNIST y con los datos que nos han salido, vamos a elegir la L2 como la mejor implementación con PCA para los datos de validación, ahora bien, tenemos que ver la ejecución en el dataset de de test de MNIST para comprobar si de verdad las distáncias son tan buenas como lo demuestran en el validation_set. Después de ejecutar el pca+knn-eva.m con las distintas distancias con y sin la aplicación del PCA, obtuvimos los datos mostrados en la tabla anterior. Podemos observar que la distancia L0 sin usar PCA se queda con un error muy grande, 17.42%, pero usando PCA se acerca al nivel de las otras distancias, 3.73% , aunque como es peor en 1% que la L3+PCA, no la elegiremos para clasificar MNIST en un principio. En cambio, si antes teniamos que la L2 era mejor que la L3,que eran mejores que la L1, ahora tenemos que la L3 es mejor que la L2,que son mejores que la L1, por lo tanto, podemos demostrar de nuevo que las implementaciones de L2 y L3 son mejores que la L1, pero también, que son parecidas entre ellas dos, ya que añadiendo un margen de error referido al procesamiento de datos nuevos, sus diferencias siguen estando cerca (0.13%), por lo que elegiremos la L3 como la mejor opción observando el test data, pero con una implementación buena, los dos serían perfectamente validos ya que el error es mínimo. Sin duda ninguna son los dos mejores al respecto a la L1, asi pues L3 es casi 1% mejor que la L1. Si observamos los datos obtenidos sin aplicar PCA, siguen el razonamiento igual.

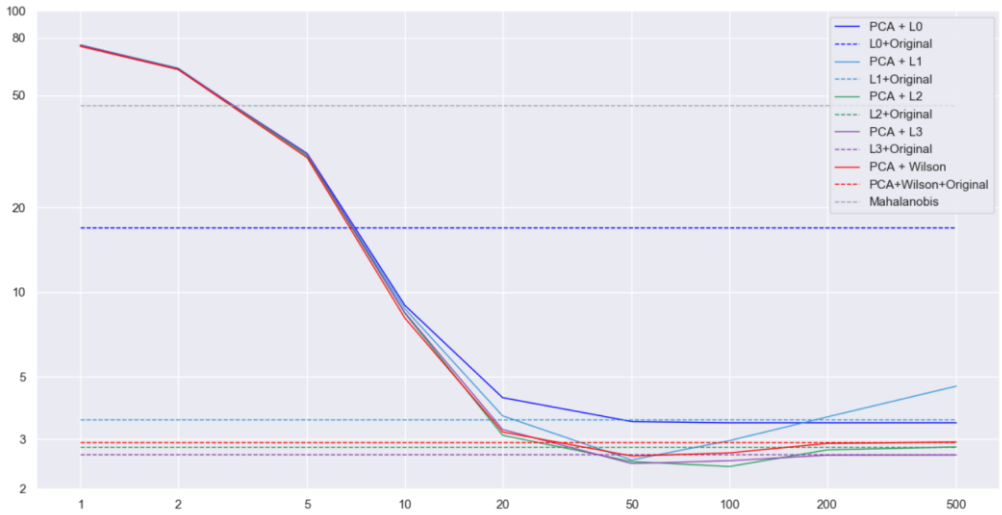


	PCA + L0	PCA + L1	PCA + L2	PCA + L3
Original (no PCA)	16,9667	3,51667	2,81667	2,65
ks= 1	75,4667	75,4666	75,466	75,466
ks= 2	61,8667	62,53	62,15	62,1166
ks= 5	31,067	30,51667	30,483	30,933
ks= 10	8,9667	8,71667	8,4	8,45
ks= 20	4,2167	3,633	3,1	3,25
ks= 50	3,467	2,533	2,5	2,46
ks= 100	3,4334	2,9667	2,4	2,5166
ks= 200	3,4334	3,6	2,75	2,6333
ks= 500	3,4334	4,633	2,8167	2,65
TEST DATA (PCA)	3,73	3,4	2,84	2,71
TEST DATA (no PCA)	17,42	3,69	3,09	2,83

***

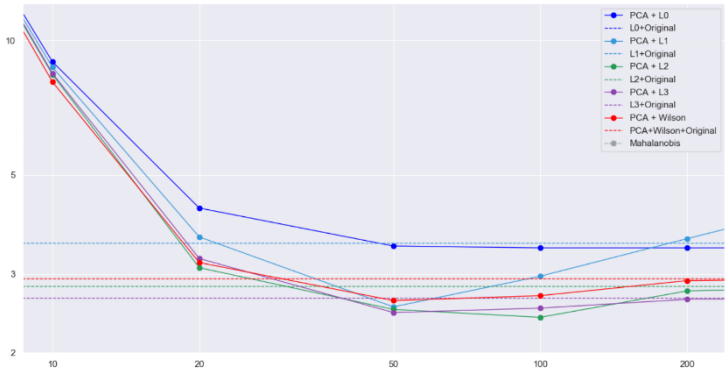
El siguiente ejercicio que estudiamos, fue la implementación de edición de Wilson, la que consiste en un preprocesado del dataset intentando eliminar todos los datos que no pertenezcan a su clase. Para

ejecutar el Wilson, tuvimos que hacer algunas modificaciones en el código, primero y antes que nada, íbamos a probarlo con y sin PCA, con lo que llamamos a Wilson antes de PCA en el pca+knn-exp.m. Dicho algoritmo nos devolvería una matriz de índices del dataset preprocesado, que nos serviría luego para calcular lo demás necesario del dataset llamando solo los datos necesitados para ello usando ese vector de índices. Nótese que así lo que conseguimos es ahorrar un montón de cálculos innecesarios por el medio como eliminaciones de datos, puesto que en cada comprobación de los datos no eliminaríamos todas las filas de datos respectivas, sino que simplemente en su posterior uso no las contemplariamos para hacer los cálculos. Luego, dentro del Wilson tuvimos que implementar varios métodos más para optimizar el código, estos son: mnn.m que nos sirve para calcular el vector de distancias de los vecinos más cercanos, y así pues no estar recalculándolos cada vez. Luego también utilizamos el knnV.m para obtener la clase del vecino y así comprobar si luego pertenece a la clase original o no. Vistos por encima los módulos que forman el Wilson, prosigamos con los experimentos, en primer lugar, vamos a ver la comprobación de los resultados con y sin PCA de Wilson con los demás sacados anteriormente, para ello usaremos la siguiente gráfica y una tabla de resultados:



En primer lugar, observamos que la implementación de Wilson sigue una curva parecida a la de L2 y L3, lo que es extraño es que su concavidad se parece más a la L3, sin embargo, para el cálculo de Wilson utilizamos la distancia L2. Por otra parte, vemos que Wilson con PCA converge posteriormente con los datos del error obtenidos con el Wilson por defecto, ya que sigue teniendo una cierta tendencia de L2 en su cálculo que le influye.

Si acercamos un poco más la vista de la gráfica en el punto en el que se intersecan las distintas líneas del error, podemos observar que Wilson sigue teniendo un error mayor respecto a los otros datos, lo que nos lleva a pensar que no es muy valido para el caso del MNIST, ya que recordemos que no tenemos que juzgar el método por su error dado en un dataset particular. Eso si, lo que hemos sacado es que la implementación de Wilson no da el mejor resultado para el dataset de MNIST. Comparándolo con los demás de nuevo, vemos que solo mejora la distancia L3 y la L1 en K=20, en dicho punto, mejora a la L1 en casi 1%, que esto es una mejora considerable, en cambio respecto a la L. no es muy distinta, por lo tanto si, es mejor en este punto, pero podría ser una casualidad de su aplicación, ya que recordemos que Wilson elimina los puntos que no cumplan una condición, por lo tanto es muy dependiente de la permutación del dataset de entrenamiento con el que lo usaremos, ya



que dependiendo de como se pongan las filas de los datos en la matriz de entrenamiento, podría dar uno u otro error. Por lo tanto, en K=20, la diferencia de error no es muy relevante, igual explicación tenemos con respecto al error de la L2. En el K=50 Wilson llega a su mínimo, por lo tanto, podemos observar que mejora las implementaciones sin PCA de las demás distancias e iguala a la implementación original de la L3, con lo que si vemos el error mostrado en la tabla de abajo, se podría suponer que los errores con Wilson son iguales entre todas las demás implementaciones, así que no

	PCA + L0	PCA + L1	PCA + L2	PCA + L3	PCA + Wilson
Original (no PCA)	16,9667	3,51667	2,81667	2,65	2,9333
ks= 1	75,4667	75,4666	75,466	75,466	74,6333
ks= 2	61,8667	62,53	62,15	62,1166	61,7666
ks= 5	31,067	30,51667	30,483	30,933	30,01667
ks= 10	8,9667	8,71667	8,4	8,45	8,08333
ks= 20	4,2167	3,633	3,1	3,25	3,18333
ks= 50	3,467	2,533	2,5	2,46	2,61667
ks= 100	3,4334	2,9667	2,4	2,5166	2,6833
ks= 200	3,4334	3,6	2,75	2,6333	2,9
ks= 500	3,4334	4,633	2,8167	2,65	2,933
TEST DATA (PCA)	3,73	3,4	2,84	2,71	2,75
TEST DATA (no PCA)	17,42	3,69	3,09	2,83	3,21

usaríamos dicha implementación para K=50 ya que es muy costosa comparada con las demás para conseguir los mismos resultados. Así pues, pasamos la implementación con Wilson por el dataset de test utilizando para ello el pca+knn-eva.m y, los resultados han sido alucinantes, ya que hemos mejorado el error respecto a la distancia L2, la cual está contenida en el calculo de Wilson, por ello pues, Wilson mejora casi un 0.1% al respecto a la distancia L2. Esto nos lleva a que a la hora de elegir un modelo, elegiríamos antes Wilson que la L2 en un principio, aunque como el error es tan pequeño, el veredicto final dependerá de las necesidades que tengamos en calculo, ya que como dijimos antes, todo el error es relativo, es decir, un modelo que va perfecto para el conjunto de testeo, podría ir muy mal para los datos de producción y al revés. Lo que respecta a las otras distancias, tenemos que mejoramos un montón frente a la distancia L1, casi 1%, en este caso si que elegiríamos antes implementar Wilson que L1 con PCA. Por último, comparándolo con la L3, vemos que empeoramos un 0.04%, por lo que dicho error se podría decir que es un 0, pero de todos modos, para usar un modelo para clasificar un conjunto parecido al MNIST, daríamos preferencia al L3+PCA que a Wilson. Resumiendo, Wilson con PCA da buenos resultados, incluso mejora la distancia L2 y L1, y se queda muy cerca para mejorar la L3. Así que el orden de elegir entre los distintos modelos sería como el siguiente, primero cogeríamos L3, luego la implementación con Wilson, a continuación, la implementación con L2 y por último la de L1. La distancia L0 deberíamos descartarla directamente, pues da un error superior al 90%. Todas las distancias mencionadas en este ranking son usando el PCA. Si observamos los errores sin usar PCA, vemos que la implementación con la distancia L2 sigue siendo mejor que Wilson, y esto es lógico, puesto que Wilson internamente usa L2. Por lo demás el ranking sigue proporcional. Así pues, para acabar volvemos a remarcar que todos los clasificadores son buenos, y que para el caso del MNIST y que tengamos que clasificar cosas relacionadas con el, usaremos el orden de preferencia mencionado anteriormente, pero luego deberíamos plantearnos que modelo será mejor para nuestro caso en concreto y la forma de utilizarlo, ya que dependiendo de la complejidad que nos podamos permitir, los modelos no se diferencian por mucho error y servirían por igual.

***

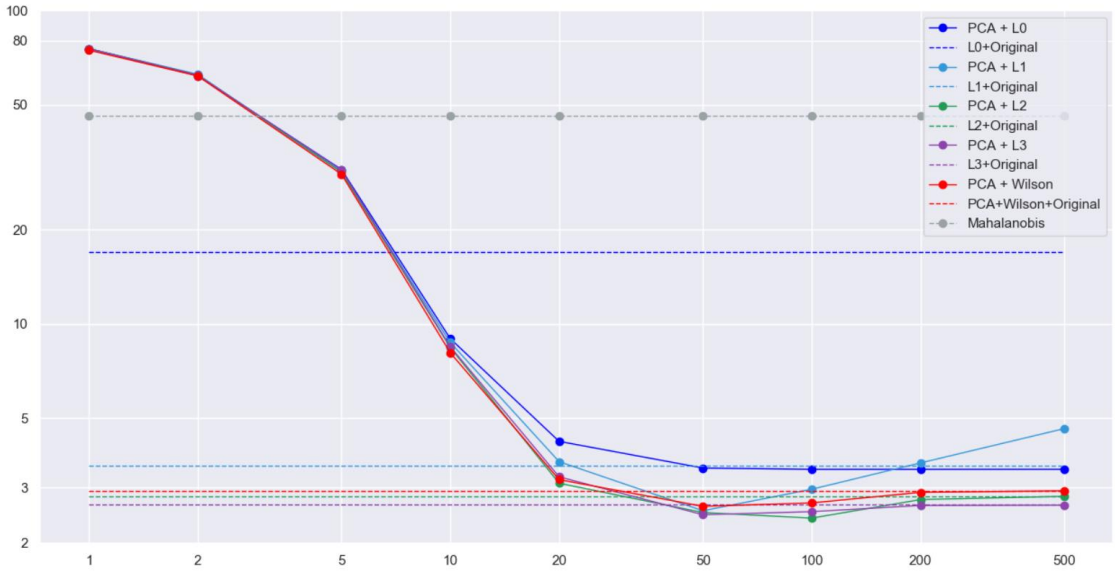
Por último, nos pusimos a implementar el ejercicio de la distancia de Mahalanobis, en primer lugar, teníamos que conseguir una formula optimizada para el cálculo del mahalanobis, lo primero que intentamos, es implementar la siguiente fórmula:

$$\mathcal{D}_{\mathbf{M}}(\vec{\mathrm{x}}_\mathrm{i},\vec{\mathrm{x}}_\mathrm{j}) = (\vec{\mathrm{x}}_\mathrm{i} - \vec{\mathrm{x}}_\mathrm{j})^\top \mathbf{M} (\vec{\mathrm{x}}_\mathrm{i} - \vec{\mathrm{x}}_\mathrm{j})$$

Su implementación debería ser en forma matricial, pero después nos dimos cuenta de que para matrices no cuadradas no nos servía, por ello volvimos a la fórmula básica ineficiente implementada en clase:

$$d(\mathbf{y},\mathbf{p}) = \sqrt{\left(\sum_{i=1}^{D} \frac{1}{\sigma_{ic}^2}(y_i - p_i)^2\right)}$$

La cual pasamos ejecutando más de 12 horas, y el único error dado fue 46.1% sin utilizar PCA. Mostrando dicho numero en la grafica junto a los demás datos sacados en las implementaciones de las demás distancias tenemos:



Como se puede observar, el error original del Mahalanobis da muy alto, por lo que cuando se utilice PCA, al final debería de pasar cerca del error y converger en el como lo hizo en los demás casos de distáncias, ya que recordemos que para su cálculo está normalizando los datos dividiéndolos entre la varianza de cada clase, por ello, cada vez el error debería tender a disminuirse conforme le vamos dando más datos de muestreo, y al final debería tender a un dato normal. Mientras se ejecutaba Mahalanobis, nos pusimos a investigar por internet sobre como podríamos optimizar nuestra fórmula, donde nos encontramos con un paper que nos daba un error del algoritmo para el entrenamiento y el test sobre MNIST, el error en el test daba 1.72%. Al ver este error, aplicando PCA, creemos que el error obtenido por nosotros para la iteración para todos los datos también está mal , ya que es de casi 50%, y como también nos quedamos sin tiempo para ejecutarlo, se dejo en suspenso. La implementación del código ineficiente de Mahalanobis está en la carpeta del código. Actualmente estamos trabajando en la función eficiente.