Universidad de Buenos Aires

Departamento de computación

Laboratorio de Datos
Trabajo práctico 02

Objetivo

Selección de modelos

Presentado por

Santiago De Luca - Lautaro Aguilar - Federico Bourrat

Grupo

“La T y la F”

Profesores

Pablo Turjanski - Manuela Cerdeiro - Mateo Guerrero Schmidt

Año

2025

Buenos Aires - Argentina

Introducción

El objetivo de este trabajo es diseñar y entrenar modelos de clasificación. Los modelos deberán ser capaces de clasificar correctamente dígitos de 0 a 9. Para ello, se dispone de una base de datos que indica, para cada imágen de 28 x 28 píxeles, el valor de la intensidad del color de cada píxel.

Primeramente entrenamos un modelo binario que tenía como propósito diferenciar entre ceros y unos. En este modelo se utilizó como método central K-Nearest Neighbors, que es un método de aprendizaje supervisado en el que se toman los K valores más cercanos al que se desea clasificar y se le asigna la clase mayoritaria.

Luego, entrenamos un modelo de clasificación multiclase en el que buscábamos clasificar cualquier dígito. Esto lo hicimos utilizando árboles de decisión. Este resultó ser el un problema más difícil, ya que la cantidad de clases a diferenciar eran muchas más que en el caso binario.

El resultado final del trabajo fue influenciado por estas características propias de los problemas a resolver y obtuvimos mejores rendimientos en el caso binario que en caso multiclase.

En este informe detallamos todos estos temas y los organizamos de la siguiente manera (links):

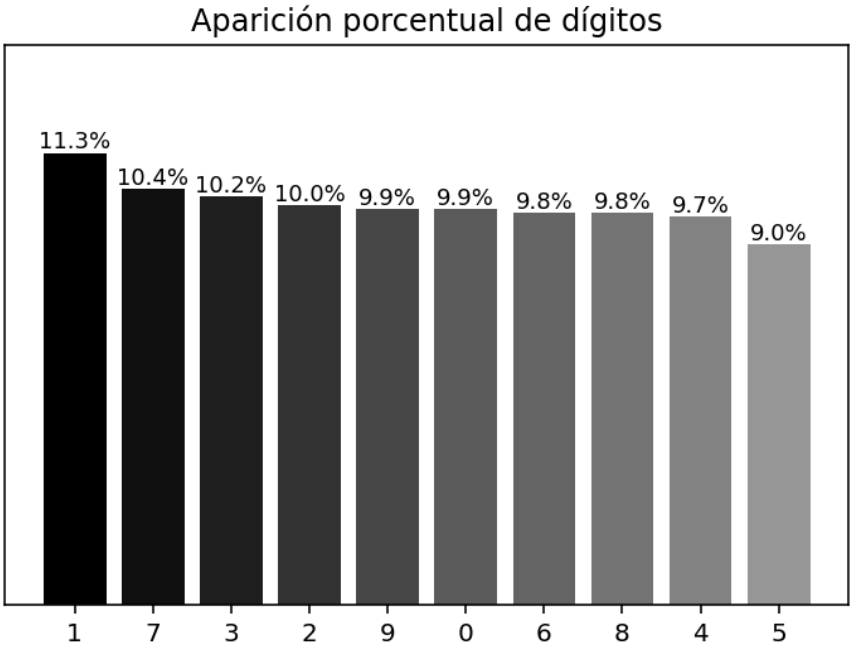
Análisis exploratorio

Realizamos un análisis de la base de datos, que es una tabla que contiene 786 columnas y 70.000 filas. Todos los valores de las celdas son números enteros. Las imágenes representadas por las filas son de 28x28 pixeles, es decir, 784 píxeles. La primera fila es un identificador único para cada imagen y la última es el dígito que forman los píxeles de esa fila.

La variable de interés a explicar es el número que cada fila representa, y son números del 0 al 9, es decir, la variable de interés toma una de 10 clases. Los valores que representan los píxeles están en un rango de 0 a 255, que indica la intensidad del color.

Los atributos relevantes para predecir el número representado son los valores de los píxeles. Por lo tanto, las 784 columnas son relevantes. La última columna también es importante ya que nos indica qué dígito representan los píxeles. Eventualmente, si se desearía eliminar algún conjunto de píxeles se podrían eliminar los de los bordes, que no aportan tanta información en la mayoría de los casos porque los números están dibujados relativamente en el centro. Esta suposición luego se verá respaldada por análisis más en profundidad de los píxeles.

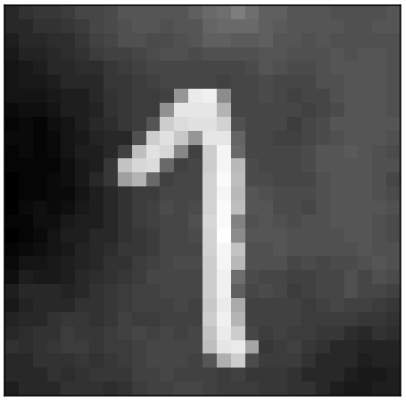
A continuación confeccionamos un gráfico para representar la aparición porcentual de cada dígito en el dataset.



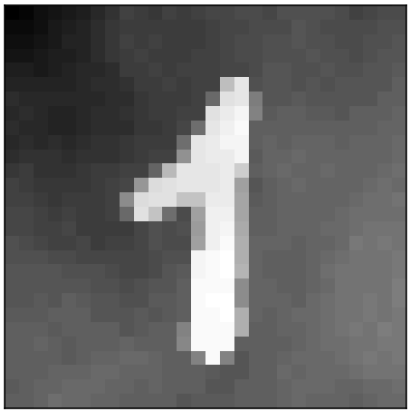
Es importante notar cómo el 5 es el número menos frecuente en el dataset. Esto traerá consecuencias al momento de entrenar modelos, ya que estos tendrán menos ejemplares del número 5 para aprender acerca de este dígito. Por otro lado, el 1 es el dígito más común.

Notamos que hay números que están escritos mucho más claros que otros, A continuación mostramos un ejemplo de un número 7 (imagen de la izquierda) que se parece mucho a un número 1. La imagen de la derecha sí es un 1, y es casi idéntica.

Clase : 7
ID: 212

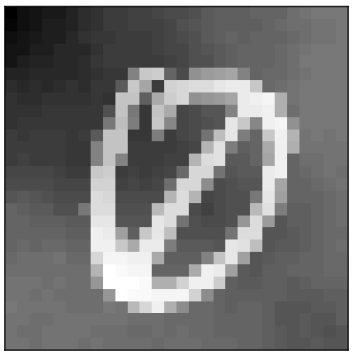


Clase : 1
ID: 8464



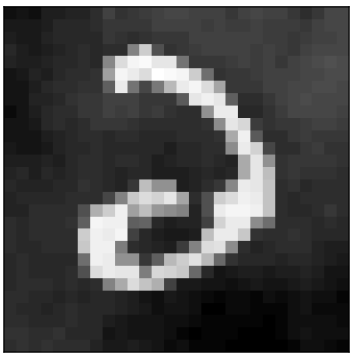
Casos como el anterior hay muchos, pero por sobre todo hay números que nos parecen indescifrables, y no llega a ser del todo claro a qué clase representan. En otros casos, hay formas de escribir números que son extrañas o poco comunes. A continuación dejamos algunos ejemplos.

Clase : 0
ID: 26728



Clase : 0
ID: 29345

Clase : 2
ID: 5656

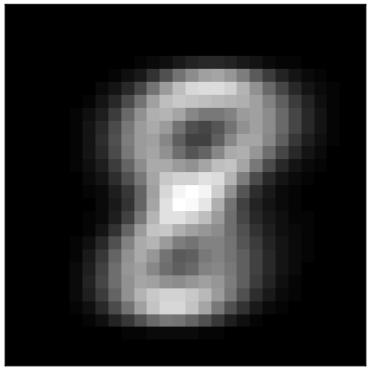


Es interesante ver cómo dentro de una misma clase, en este caso el 0, hay números con formas extrañas y diferentes entre sí. En el caso del 2, creemos que aún viéndolo como humanos no podríamos decir con certidumbre qué número representa.

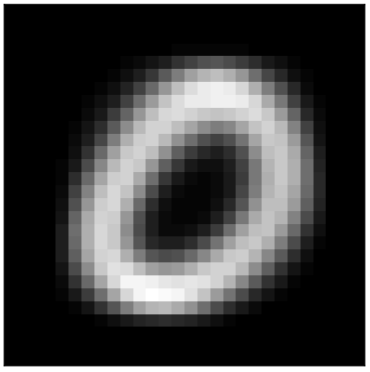
De todas formas y, a pesar de haber algunos casos difíciles de clasificar correctamente, en general se puede ver que la mayoría de dígitos de cada clase sí tienen una forma esperada.

Con el objetivo de profundizar el análisis, calculamos para algunos dígitos los promedios de todos sus píxeles y obtuvimos como resultado imágenes que llamamos “dígitos promedio”. Aquí graficamos algunos (escala de grises, blanco es 255 y el negro 0).

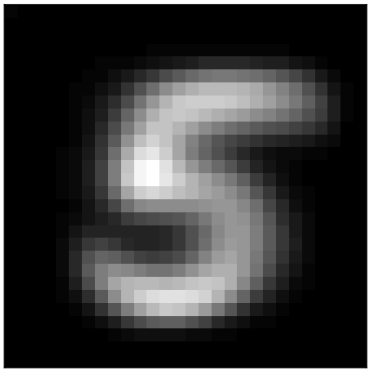
8 promedio



0 promedio



5 promedio



Notar como los píxeles por los que más se suele trazar al dibujar un número son más claros y cómo los bordes se ven difuminados. También se puede ver cómo los píxeles menos relevantes parecen ser aquellos que están en los bordes, porque casi siempre son completamente negros. Eventualmente, estos se podrían eliminar, reduciendo así el tamaño de la imagen. De igual manera se nota la importancia de los píxeles centrales.

A diferencia de otros datasets, las columnas que tenemos únicamente representan la intensidad de un píxel en cierta posición. Estos valores no tienen ninguna otra semántica asociada, ni ninguna otra propiedad, dificultando su manejo. Un punto central del trabajo es cómo extraer información de ellos y cómo determinar qué píxeles proveen más información que otros.

Clasificación binaria

Análisis previo

Para analizar el conjunto de datos del dataset binario se aislaron y se contaron los unos y ceros. Se encontraron 6903 ceros y 7877 unos, es decir, el 46,7% de los dígitos del dataset en su versión binaria eran cero. Esto nos indica que este está relativamente balanceado en cuanto a ceros y unos.

De esta información se puede suponer que un modelo que elija valores al azar obtendrá, la mayoría de veces, un 50% de aciertos aproximadamente. Esto quiere decir que un modelo con esa performance o levemente menor debe ser descartado ya que, a pesar de que el número parezca alto, el modelo no hace buenas clasificaciones.

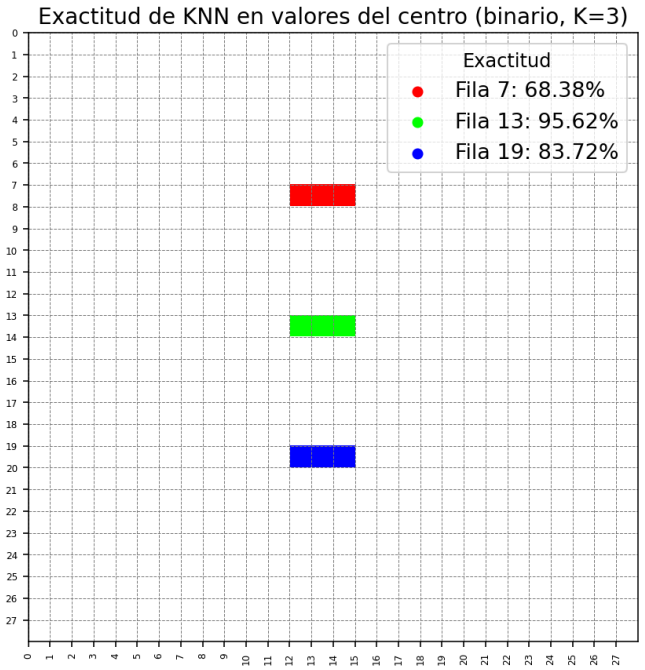
Por otro lado, es importante notar que los píxeles de la base de datos se encuentran entre 0 y 255 y represententan intensidades de color, es decir, están en la misma escala. Ya que el modelo a utilizar en esta sección es K-Nearest Neighbors, es esencial recalcar que no hace falta reescalar los datos porque estos se encuentran en la misma escala.

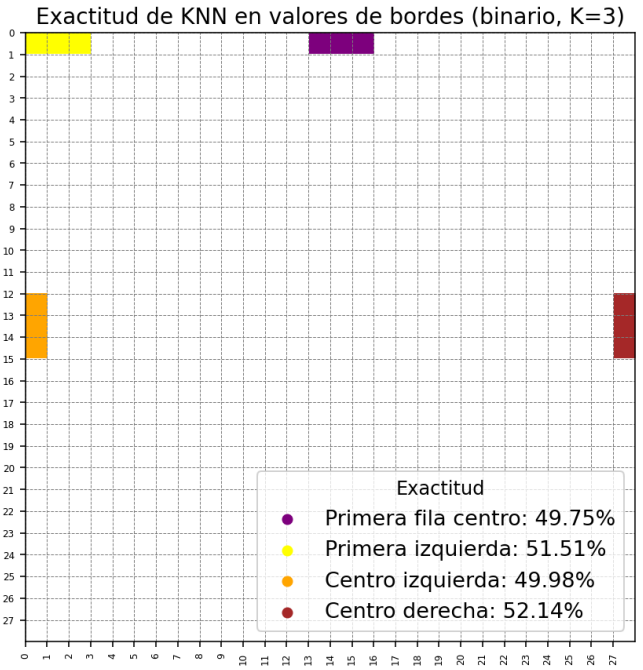
Primeros acercamientos al modelo

En un primer acercamiento al modelo probamos entrenar un modelo de K-Nearest Neighbors (KNN) con K = 3. Para elegir sobre qué píxeles entrenar, probamos en un principio con solo tres también. Entrenamos primero sobre píxeles centrales, ya que supusimos que iban a dar más información acerca de la diferencia entre el dibujo del 0 y del 1, porque la mayoría de los números están centrados. La mejor exactitud obtenida fue de 95.64% (por comodidad expresaremos la exactitud en forma de porcentaje en vez de su rango normal entre 0 y 1). Esta exactitud surge al entrenar sobre los tres píxeles centrales de la imagen (de color verde en el gráfico).

Luego probamos eligiendo píxeles de los bordes, esperando peor performance. Esto fue así, siendo la mejor exactitud de 52.14% (de color marrón en el gráfico), lo cual es muy bajo dado lo explicado en el análisis previo.

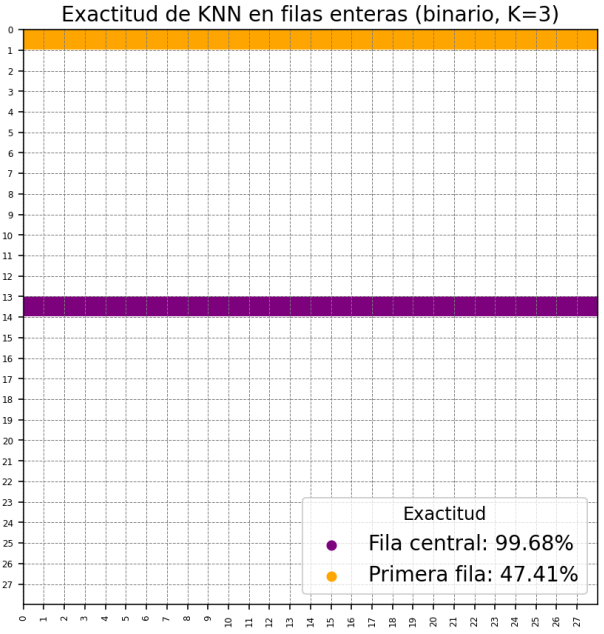
A continuación mostramos los gráficos de los píxeles elegidos para las pruebas, junto a su performance. Cada grilla de 28x28 representa una imagen.





Es importante recalcar que cualquiera de los tríos de tuplas centrales tiene mejor performance que los de los bordes. El peor de los centrales tiene 68% de exactitud, mucho mejor que los de los bordes.

Para intentar mejorar la performance obtenida, elegimos probar con más píxeles. Probamos entrenando KNN con K = 3 para los 28 píxeles de la primera fila, y luego para los 28 de la fila central. Como era esperado, la performance sobre la fila central fue mejor que la de la primera fila, y mejor que la performance de solo tres píxeles de la fila central. Esta exactitud fue de 99.68%.



Vale la pena notar la pésima performance del modelo entrenado sobre la primera fila (47%), a pesar de ser un modelo entrenado sobre 28 píxeles. Los modelos que antes entrenamos sobre solo 3 píxeles centrales fueron mejores.

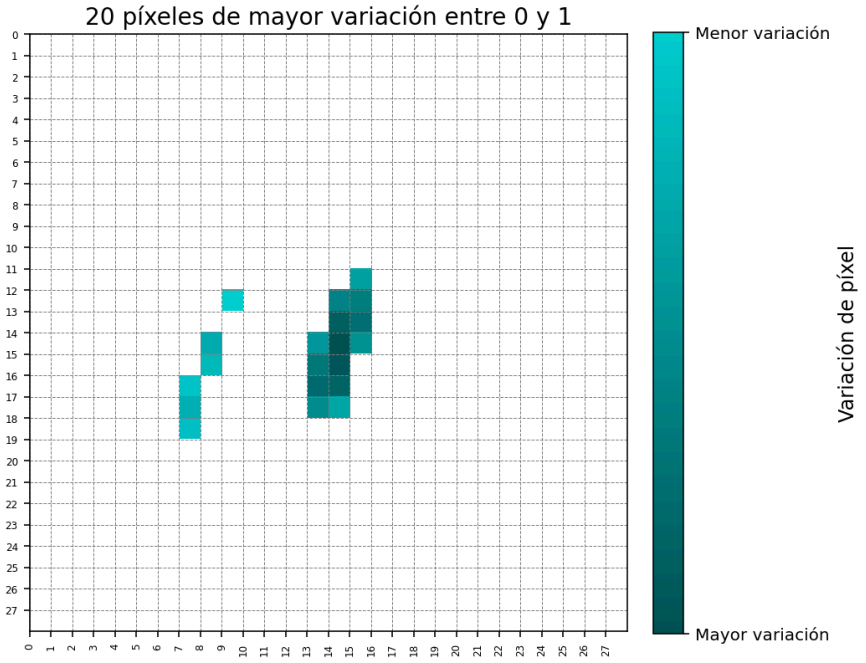
Aunque la performance sobre la fila central fue muy buena, sabíamos que se podía conseguir mejor exactitud ya que, en esta selección de 28 píxeles se están incluyendo píxeles que ya vimos que brindan poca información, que son los píxeles que están sobre los bordes.

Elección del modelo

Para decidir cuál era el mejor modelo, tuvimos que encontrar la mejor cantidad de píxeles y cuáles eran estos, y el mejor valor de K. Para encontrar los mejores píxeles calculamos el cero promedio y el uno promedio, de la forma mencionada en el análisis exploratorio, y luego calculamos la diferencia entre cada uno de los píxeles de ellos. Así obtuvimos los píxeles en los que, en promedio, el valor varía más entre los ceros y unos. Las diferencias más grandes nos indican que ese píxel otorga más información para diferenciar entre los dígitos.

Entonces probamos entrenar modelos con K de 1 a 12 y, para cada uno de los K, probamos con entrenar el modelo de 1 a 25 píxeles con mayor variación entre cero promedio y uno promedio (entrenamos 300 modelos en total). De esta forma, obtuvimos una grilla con todas las performances de cada modelo dependiendo de los hiperparámetros observados. La mejor exactitud fue de **99,81957%**, en el modelo que tomaba los 20 píxeles con mayor variación y tenía K = 3. Notar cómo este modelo obtuvo mejor exactitud usando 8 píxeles menos que el modelo entrenado sobre la fila central, que usó 28 píxeles y tuvo exactitud de 99,68%.

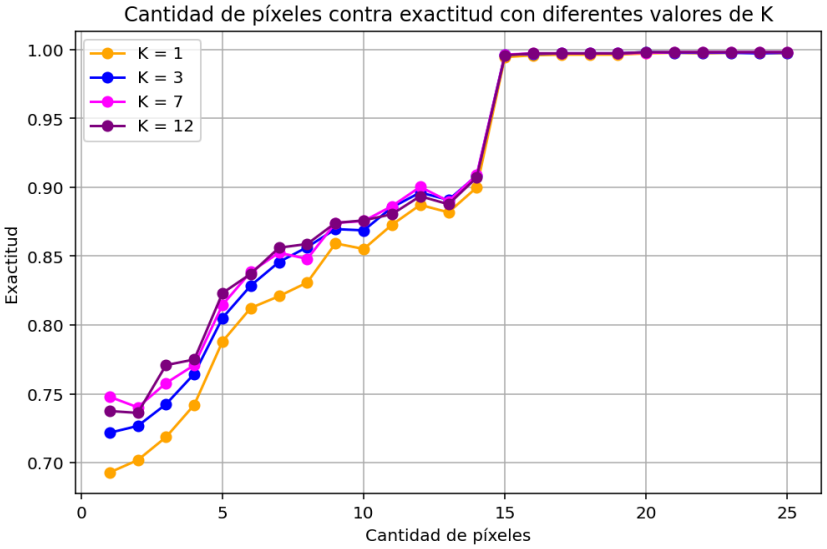
A continuación graficamos los 20 píxeles con mayor variación sobre los cuales se entrenó el modelo final. Los píxeles más oscuros son los que más varían, los más claros son los que menos.



Como era de esperar, los píxeles de mayor variación entre el cero promedio y el uno promedio son los del centro del trazo del cero, donde la mayoría de ceros no tienen nada dibujado, pero los unos sí. Esto explica las altas exactitudes obtenidas cuando probamos con tres píxeles centrales.

Luego, otros píxeles con mucha variación son los que corresponden a la panza izquierda del cero, donde en la mayoría de trazos de unos los valores son bajos (no se escribe sobre ese lado de la imagen), y en los ceros altos.

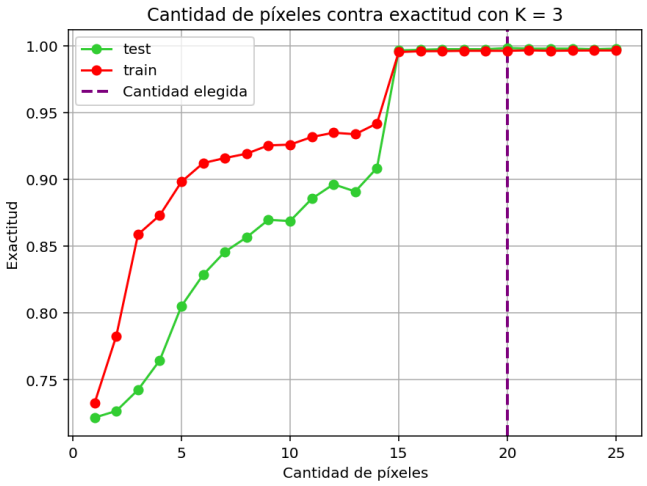
Para entender cómo evoluciona la performance con los diferentes hiperparámetros, confeccionamos el siguiente gráfico.

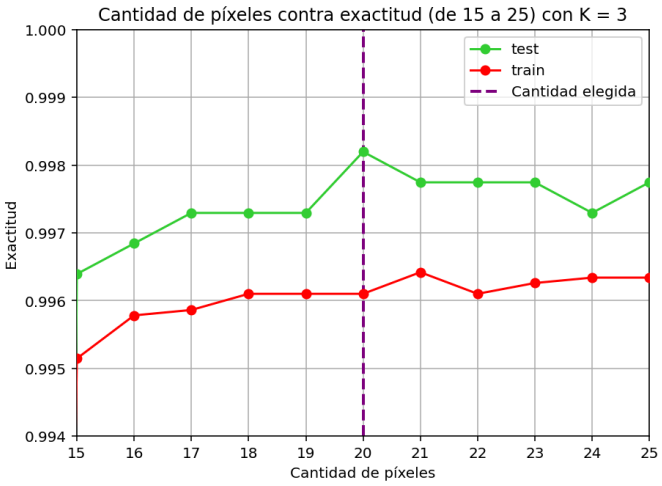


En este gráfico es fácil notar que hay un crecimiento importante de la performance a medida que la cantidad de píxeles aumenta, para todos los K, hasta que la cantidad de píxeles alcanza 15. De ese punto en adelante la performance se ameseta y se mantiene prácticamente constante, aunque se ve un pequeñísimo aumento entre 15 y 20 píxeles. Las diferencias de performance dependiendo de K son pequeñas, sobre todo luego de los 15 píxeles. Lo que se nota es cómo la performance de K=1 es mala, ya que probablemente esté sobreajustando a los valores de train y entonces tiene malos resultados en test.

Análisis de sobreajuste

Para evaluar la posibilidad de que nuestro modelo esté sobreajustando, confeccionamos los siguientes gráficos.

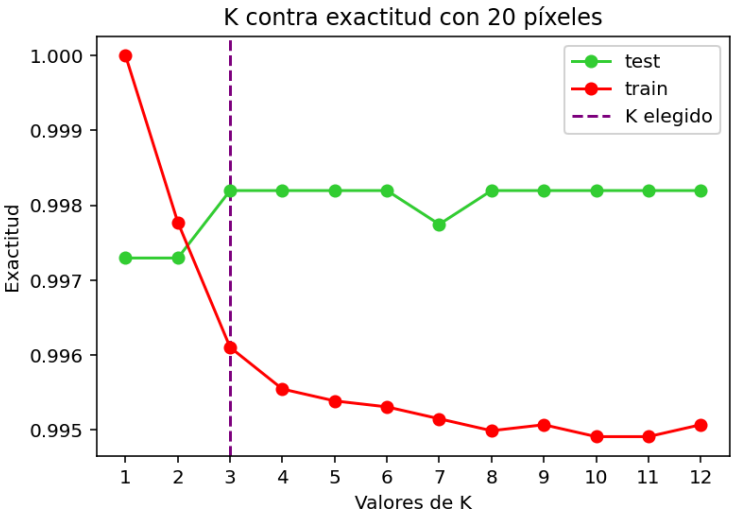




En el caso de la evaluación de la performance de train y test en los diferentes modelos, se nota también un punto central de inflexión cuando el número de píxeles 15 es elegido. De 1 a 15 la performance en train es mejor que en test, pero a partir de 15 píxeles la performance en test pasa a ser levemente mejor que la de train, mientras ambas se amesetan. Entre esas exactitudes de test elegimos la mejor, que fue en 20, punto en el que la performance en test también está sobre la de train.

Esto es un buen indicio de que no estamos sobreajustando, ya que la performance en test del modelo elegido es mejor que la de train. De todas formas, vale aclarar que ninguno de estos análisis nos permite decir contundentemente que no estamos sobreajustando, sólo nos ayudan a encontrar posibles señales de sobreajuste.

Acerca del sobreajuste en K hicimos el siguiente gráfico, donde lo que dejamos fijo ahora es la cantidad de píxeles (en 20). Es importante leer este gráfico teniendo en cuenta que KNN sobreajusta para valores chicos de K.



De nuevo es bueno notar que para el valor de K elegido la performance sobre test es mejor que en train. Si hubiéramos elegido K = 1 o 2 sí tendríamos un problema, porque tendríamos mejor performance en train que en test. También es llamativo cómo la exactitud queda prácticamente estancada a partir de K = 3. Esto es bueno porque no se ve en el gráfico una mejor performance en test que la del modelo que elegimos, es decir, no hay un K con el que tendríamos mejor performance en test.

Clasificación multiclase

Decisiones previas

El modelo que utilizamos en esta sección fue un árbol de decisión. En el caso de KNN, es importante qué atributos se le dan para que entrene, ya que KNN no realiza ninguna decisión sobre qué atributos son mejores para diferenciar clases. En el caso de árboles de decisión esto es muy distinto, ya que una parte central del algoritmo de entrenamiento de los árboles de decisión es que ellos eligen el mejor atributo sobre el cual tomar la decisión.

Es por esto que, al momento de entrenar los árboles, les dimos los 784 píxeles para que ellos elijan, en cada paso, sobre qué píxel tomar las decisiones. No limitamos su capacidad de utilizar los atributos.

Eventualmente, si se quisiera hacer más barato computacionalmente el proceso de entrenamiento, se podrían entrenar los árboles solo sobre algunos de los píxeles que más varían (en este caso, la variación sobre los 10 dígitos).

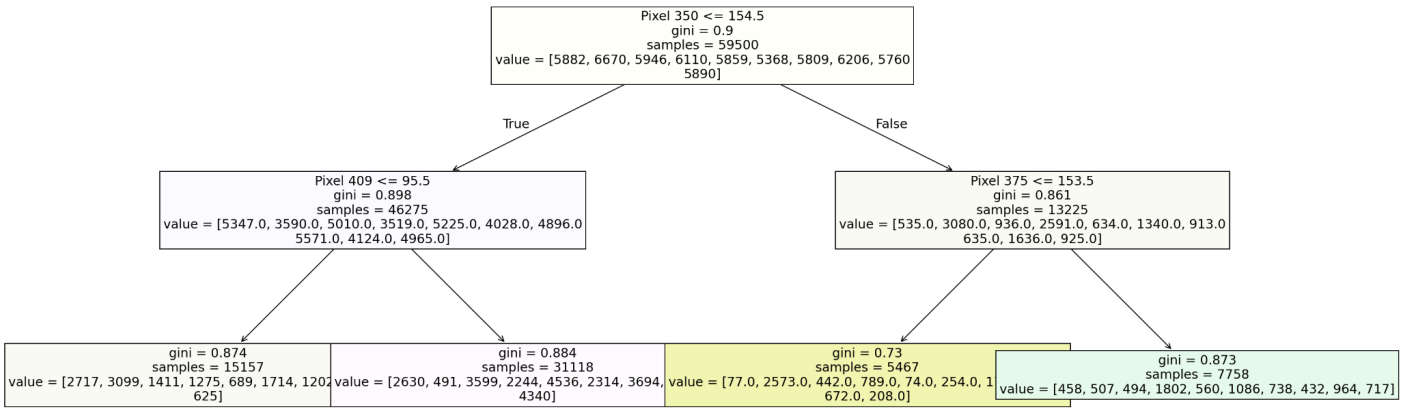
En cuanto al balance de los datos, se estudió en el análisis exploratorio cómo la distribución es relativamente pareja, a pesar de que el 5 tenga especialmente menos apariciones, y el 1 más, en comparación a los demás dígitos.

Primeros acercamientos al modelo

Inicialmente entrenamos dos árboles para observar sus performances, uno con altura máxima 2 y criterio Gini y otro con altura máxima 3 y criterio entropía.

La exactitud en el árbol entrenado con Gini fue de 20.43%. Para entender este número, es esencial tener en cuenta que, en el problema a resolver, la performance de base es de aproximadamente 10%, ya que hay 10 clases que están distribuidas a grandes rasgos de manera proporcional. Esta performance mejora ese porcentaje de base.

A continuación graficamos el árbol resultante del modelo.



Ya que hay 10 clases distintas y el árbol indica en cada nodo la cantidad de valores en cada clase, es muy complicado mantener un balance entre buena visualización y recorte de información, así que en esta imagen se superponen las hojas del árbol.

En este gráfico se ve como los tres atributos elegidos para tomar las decisiones no están tan lejos entre sí (350, 375 y 409). De estos tres, solo uno de ellos está entre los 20 de mayor variación entre 0 y 1, este es el píxel número 350. Tiene sentido que los píxeles que brinden más información sean otros porque ahora se están teniendo en cuenta muchas más clases.

Notar lo malo de los valores obtenidos en el criterio de Gini, aún en las hojas, donde el mejor valor es de 0.73. En el criterio de Gini valores cercanos a cero son mejores, indican menos mezcla de clases.

El segundo modelo entrenado fue con altura máxima 3 y entropía como criterio, y obtuvo mejor rendimiento. Su exactitud fue de 29,0%. Esto tiene sentido ya que la profundidad del árbol es mayor, pero todavía está lejos de ser un número satisfactorio.

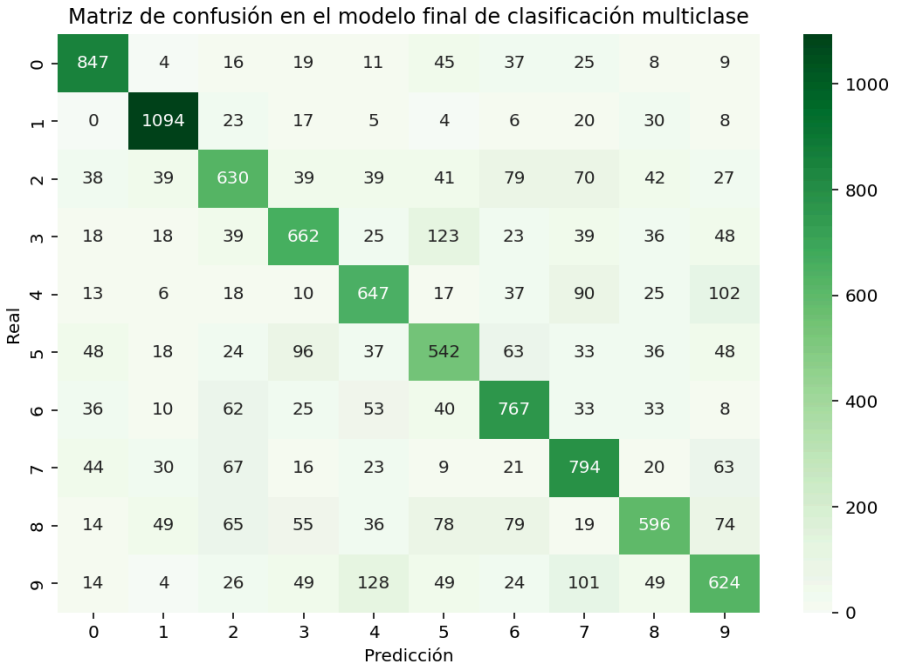
Elección del modelo

La elección del modelo final se realizó eligiendo el mejor de 18 modelos que entrenamos: probamos con 9 alturas (de 2 a 10) y con 2 criterios (Gini y entropía). Todos estos modelos fueron entrenados con K-folding y el que tuvo mejor exactitud fue el elegido. Ese árbol fue el de altura máxima 10 con entropía.

Este modelo obtuvo, sobre el conjunto de test, un promedio de exactitud de 67,5579%, que supera muy ampliamente los modelos entrenados al explorar el problema previamente. Esta performance está muy lejos del 10% de base, y supera con tranquilidad la exactitud en la mitad de los casos.

Luego, se entrenó el modelo sobre el conjunto de desarrollo entero y se lo testó sobre un conjunto *held-out* separado al inicio, sobre el cual no se entrenó ningún modelo multiclase. Sobre este nuevo conjunto, el modelo tuvo una exactitud de **68,6%** que, para nuestra sorpresa, ¡mejoró su performance promedio sobre el conjunto de test! Esta era de 67,56%, así que la superó en más de un 1%.

Para analizar esta performance graficamos la siguiente matriz de confusión.

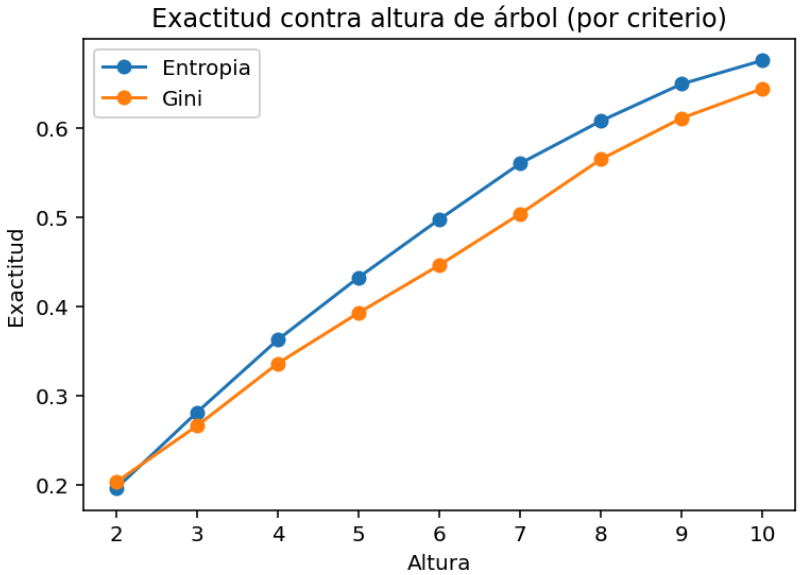


En la matriz se puede ver cómo la mayoría de los valores están en la diagonal (es decir, fueron correctamente clasificados). Es interesante notar las confusiones más comunes: el 4 y el 9 son los más confundidos, 128 veces se confunde un 9 por un 4 y 102 veces un 4 por un 9. Esto se podría explicar por su forma relativamente similar en el trazo.

Otro par muy confundido es el 5 y el 3, 123 veces se confunden 3 por 5 y 96 veces se confunden 5 por 3. Para entender esto vale la pena recordar que el 5 es el dígito con menos apariciones en todo el dataset, así que tiene sentido que sea confundido, ya que tenemos menos ejemplares para aprender su forma.

Llama la atención lo confundido que es el 9 por el 7 (101), pero lo poco confundido que es confundido el 7 con el 9 (63). Esto se podría explicar con que hay un problema especial en el trazo del 9, en el que el círculo superior queda más achatado, haciendo que el modelo lo interprete como 7.

El criterio elegido en el modelo final fue entropía. Para estudiar los criterios, confeccionamos el siguiente gráfico para analizar cómo evoluciona la performance en cada uno de los criterios a medida que las alturas varían.

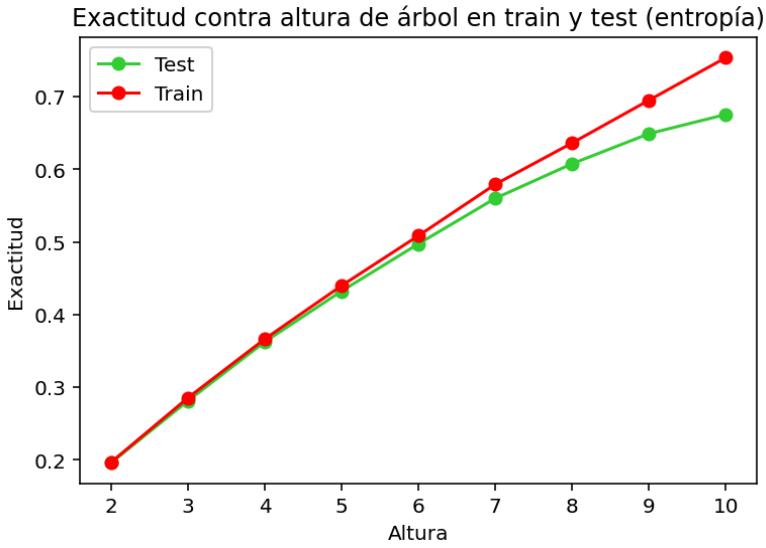


Las exactitudes graficadas son las exactitudes promedio en los conjuntos de test realizados en los K-folding.

Lo central a ver en este gráfico es cómo la entropía genera mejor performance que Gini en casi todas las alturas. Es interesante ver que la única altura en la que Gini tiene mejor performance que entropía es en la más pequeña, altura 2. También notamos que, para las últimas alturas, parecería que la performance con entropía tiende a amesetarse más rápido, sugiriendo que en alturas más grandes no graficadas, la performance sobre Gini podría volver a superar a la de entropía.

Análisis de sobreajuste

Para analizar la posibilidad de sobreajuste al entrenar el modelo, utilizamos los datos de las performance promedio en K-folding sobre test y train del criterio de entropía y armamos el siguiente gráfico.



Es importante leer este gráfico teniendo en cuenta que los árboles de decisión sobreajustan con alturas demasiado grandes. En este gráfico vemos como, en la altura elegida, existe una diferencia entre la performance en train y test, pero esta es pequeña. Como era de esperarse, la tendencia que se nota en el gráfico es que a medida que la altura crece la performance de test tiende a amesetarse rápidamente mientra que la de train crece.

Como la diferencia de exactitud es pequeña no creemos que este gráfico sea un gran indicio de sobreajuste y, sobretodo, creemos que nuestro argumento más fuerte para decir que no estamos sobreajustando es que la performance del modelo en el *held-out* superó la del promedio de performances en el test. Es decir, parecería que el modelo no está perdiendo capacidad de generalización porque tiene buena performance en datos sobre los que nunca se entrenó. Si se estuviera sobreajustando severamente, se vería una performance peor en el *held-out*. De nuevo, ninguno de estos análisis nos permiten asegurar que no estamos sobreajustando, sólo nos ayudan a entender mejor el comportamiento de nuestro modelo.

Conclusiones

El estudio realizado se puede resumir en dos números, que son los valores de las exactitudes obtenidas en cada uno de los modelos entrenados. En el caso de clasificación binaria, la exactitud final fue de 99,81957%, mientras que en el caso de clasificación multiclase la exactitud del modelo final fue de 68,6%.

Es esencial entender a qué se debe esta diferencia. En el caso de la clasificación binaria, sólo se contaban con dos clases que contenían relativamente la misma cantidad de elementos. Esto genera que la exactitud de base sea del 50%, ya que un modelo que siempre dice 0 o siempre dice 1 acierta la mitad de las veces. Para la clasificación multiclase esto fue muy distinto ya que la cantidad de clases con las que se contaba fueron 10, que también estaban distribuidas relativamente con la misma frecuencia, y esto generó que la exactitud de base sea solo de 10%.

En definitiva, un modelo mejoró una exactitud de 50% a 99,8%, mientras que otro mejoró una exactitud de 10% a 68,6%. Es decir, la diferencia de performance de los modelos finales es explicada por la dificultad de la clasificación en cada uno de los problemas que resuelven.

La conclusión que podemos extraer de estas diferencias es que el entrenamiento de modelos de clasificación se complejiza mucho al aumentar la cantidad de clases. Esta complejidad debe ser acompañada por modelos más fuertes y más entrenamiento.

En el caso de clasificación multiclase se entrenó un árbol de altura máxima 10 sobre 784 píxeles. Este entrenamiento fue mucho más complejo que el recibió el modelo final de clasificación binaria de K-Nearest Neighbors, que fue entrenado sobre 20 píxeles con K = 3. Aún así, la performance del modelo más sencillo fue mejor, porque el problema al que se enfrentaba era más simple.