Trabajo Práctico nº 2

Laboratorio de Datos

Nombre del grupo: Data Wizards

Integrantes: Valentina Morrone, Carlos Rafael Chaves Lopez, Jimena Jofré

LU: 35/24 - 19/23 - 696/23

Mails: valenmorrone@hotmail.com, rafael.chaves.lopez1@gmail.com, jimeejofee@gmail.com

Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires.
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Guiraldes 2610 -C1428EGA
Ciudad Autónoma de Buenos Aires

Introducción

En el siguiente informe realizamos un análisis del dataset MNSIT-C en su versión *Fog*. Usamos la información de sus píxeles para intentar predecir los dígitos que forman usando KNeighbors y Tree Decision Classifier.

Resumen:

En la primera parte, entrenamos un modelo de K-Neighbors que pueda diferenciar los 0 de los 1. Utilizamos tres conjuntos diferentes de vecinos para lograr un modelo final con buena accuracy. En la segunda parte, usamos todo el dataset para predecir a cuál de los diez dígitos corresponde una imagen. Utilizamos Decision Tree Classifier, consiguiendo un modelo complejo y de baja accuracy.
Concluimos en que con este dataset, el modelo KNN puede ser muy bueno para distinguir entre imágenes de 0 y 1, con una exactitud de 0.98 y que el modelo de Tree Decisión Classifier no es muy bueno para distinguir el número de una imagen, con una exactitud de 0.62.

Ejercicio 1: Análisis Exploratorio

El dataframe con el que trabajamos en este informe tiene 70.000 filas y 786 columnas, con un total de 55.020.000 datos. Cada fila conforma una imagen de 28x28 pixeles, los valores de las casillas representan el nivel de gris que tiene cada pixel de la imagen y van del 0 al 255. Los valores más altos son las tonalidades más blancas y es lo que nos permite diferenciar de qué número se trata. Los valores intermedios, son grises que hacen que la imagen tenga menos definición o sea más difícil de distinguir. Los valores más bajos corresponden al fondo oscuro de la imagen. La primera columna es el índice de la base de datos y la última es una etiqueta que nos indica qué número forma cada imágen.

a) Dado que la columna del índice no la utilizamos al momento de graficar las imágenes, no es relevante para predecir la imagen. La columna label tampoco, aunque sí para comprobar si la predicción es correcta.

Analizando las imágenes generadas por las filas del dataframe, podemos notar que los números se encuentran centrados.

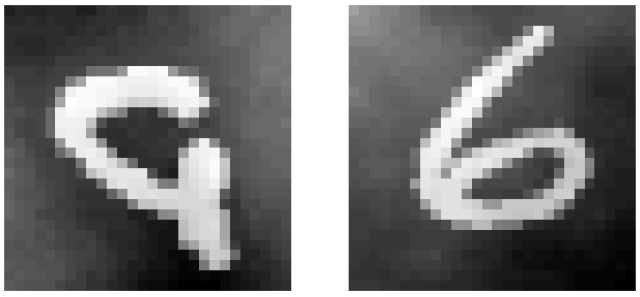

Figura I. Imágenes aleatorias del dígito 9 y 6 respectivamente.

Podemos notar que, en general, los márgenes son más oscuros, por lo que podríamos obviarlos. La información relevante se encuentra en el centro de la imagen.

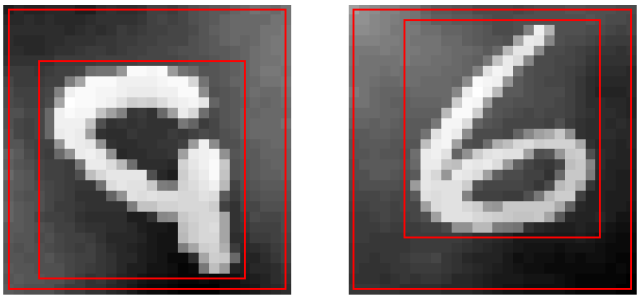

Figura II. Centralización de los números.

Para comparar mejor la distribución de la escala de grises entre píxeles, creamos 10 imágenes con el promedio del valor de cada píxel que representa su dígito.

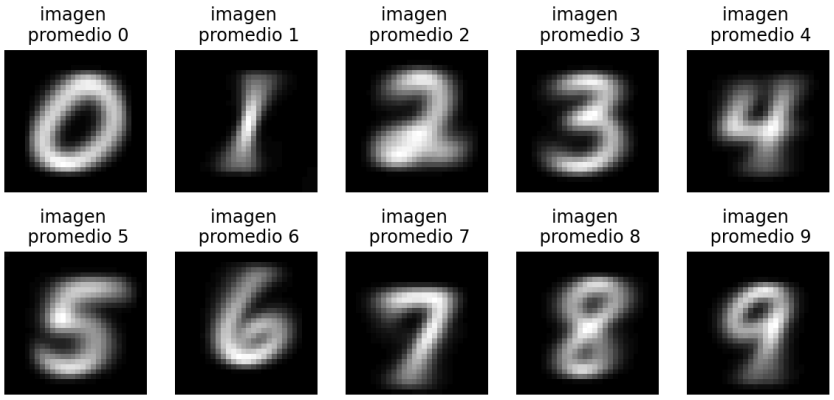

Figura III. Promedio de valores de grises de cada dígito.

Notamos que los píxeles cercanos a los márgenes se mantienen sin relevancia, es decir, no nos aportan información para determinar de qué número se trata.

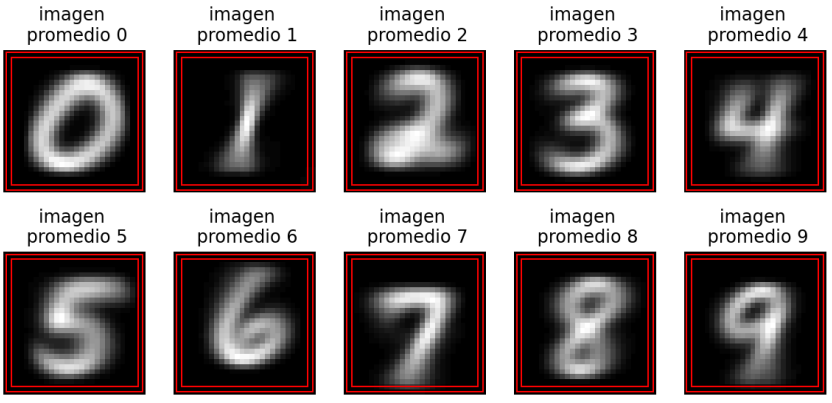

Figura IV. Centralización del promedio de grises de cada dígito.

A continuación creamos una imagen promedio juntando todos los números para comparar la distribución de la escala de grises a lo largo de todos los números.

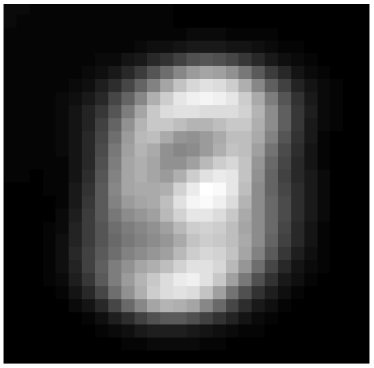

Figura V. Imagen promedio de todos los dígitos unidos

Comparamos los valores en la imagen agrupando y promediando los píxeles a lo largo del eje x e y.

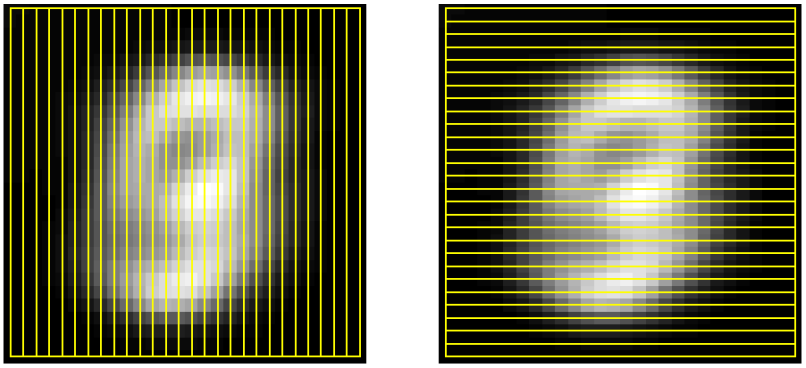

Figura VI. Imagen promedio con píxeles agrupados a lo largo del eje X (izq) y del eje Y (der).

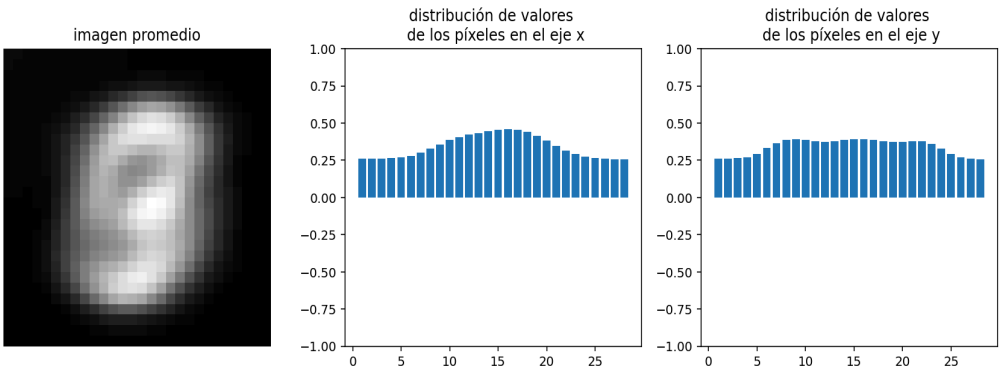

Figura VII. Distribución de los valores en los píxeles según el eje X (izq)

y el eje Y (der).

Llegamos a la conclusión a partir de este último gráfico y el análisis previo que los márgenes también podrían ser descartados. Decidimos no hacerlo puesto que no es indispensable.

b) Existen números más parecidos entre sí que otros. Es notable que el número tres es más parecido al número ocho que al número uno:

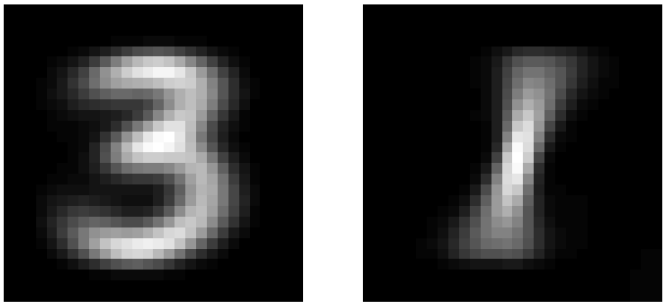

Figura VIII. Comparación del promedio de grises del número tres y el uno.

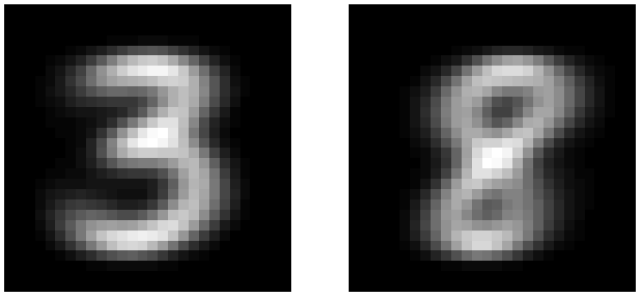

Figura IX. Comparación del promedio de grises del número tres y el ocho.

Para determinar qué números son más parecidos entre sí, restamos las imágenes promedio de ambas comparaciones y luego identificamos el rango de valores que toman en el eje X y en el eje Y.

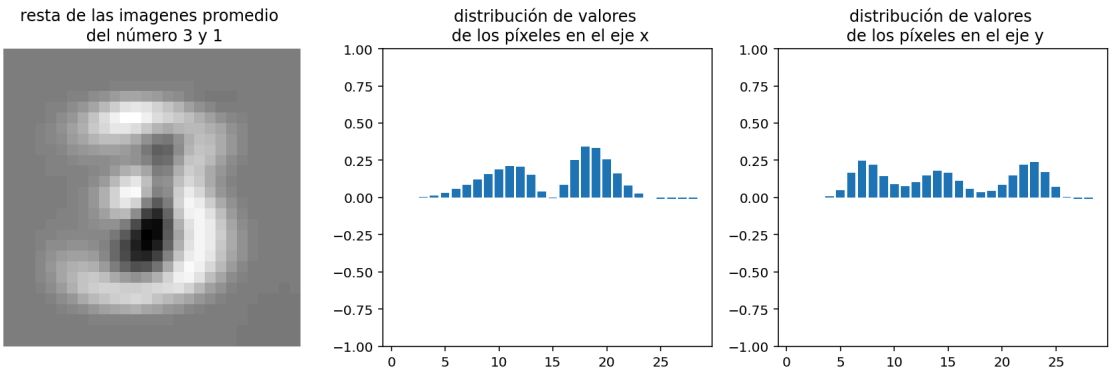

Figura IX. Resta del promedio de tres y uno con su respectiva distribución de grises en el eje X e Y.

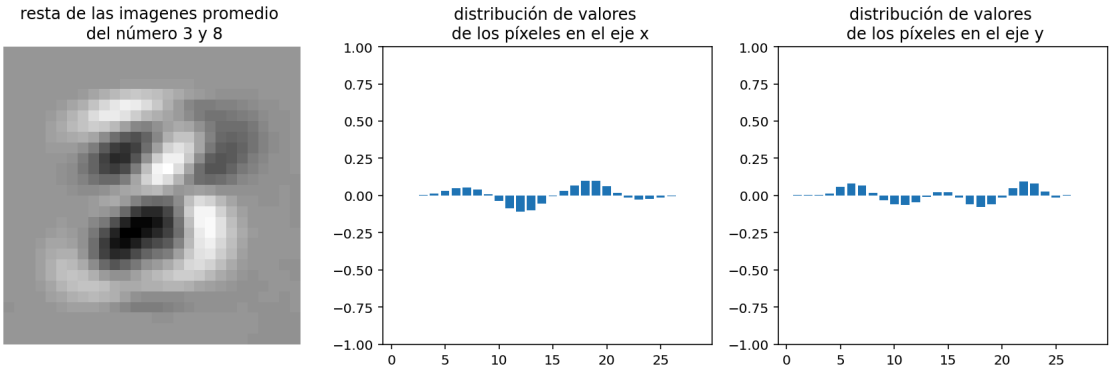

Figura X. Resta del promedio de tres y ocho con su respectiva distribución de grises en el eje X e Y.

Como podemos apreciar, el rango de valores de la resta de las imágenes promedio del número 3 y el 8 en ambos ejes es mucho más acotado al de la resta de las imágenes promedio del número 3 y 1, por lo que concluimos que el número 3 es mucho más parecido al 8 que el al 1.

c) Las imágenes presentan similitudes entre sí, aunque tienen ciertas variaciones. A veces es ligeramente diferente la forma: hay algunos más aplanados u otros más alargados. En general, todos mantienen la misma estructura y son fácilmente reconocibles, y esto se puede ver en la imagen que promedia los valores de las tonalidades del número cero. Todas esas variaciones de un mismo número, si se promedian, lo terminan dibujando de una forma bastante clara.

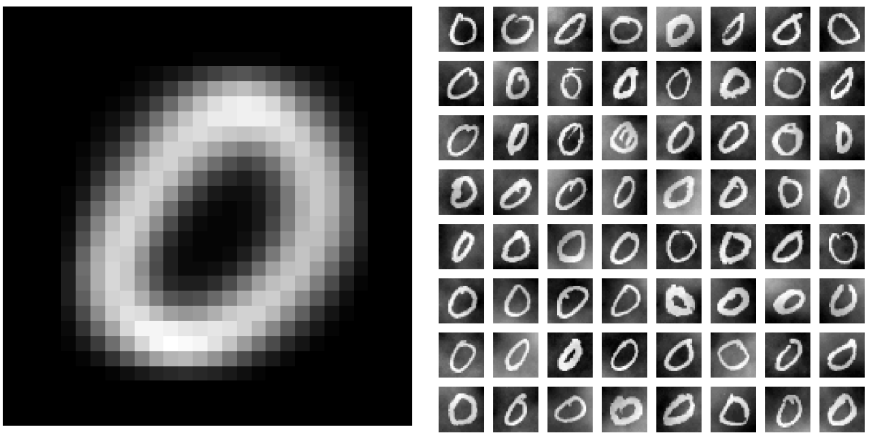

Figura XI. Comparación general de la imagen promedio del número 0 y una muestra de imágenes del número 0.

d) Con el dataset de Titanic pudimos realizar una predicción a partir de una regresión lineal simple. Este modelo es sencillo y más intuitivo porque se trataban de valores numéricos. Para el dataset de este trabajo debimos primero normalizar los valores de todos

los pixeles y generar las imágenes. Además de segmentarlas según el número del cuál se trataba y sacar el promedio de tonalidades de cada número. Además el orden de los datos importa (ya que representan el valor de cada píxel en la imagen y la imagen se vería distorsionada si no se tiene en cuenta el orden del píxel, o formará otro dígito) mientras que en los ejemplos con los que trabajamos en clase no.

Ejercicio 2: Clasificación Binaria

El objetivo de este punto es clasificar imágenes de dígitos manuscritos en dos clases: 0 y 1. Para ello, vamos a utilizar un modelo de K-Nearest Neighbors (KNN). El proceso incluye la preparación de los datos, la selección de atributos, el entrenamiento del modelo y la evaluación de su rendimiento utilizando métricas de clasificación como la exactitud *(accuracy)*.

a) Filtramos el dataframe original para conservar únicamente las imágenes correspondientes a los dígitos 0 y 1. Luego, analizamos la cantidad de muestras por clases, y obtenemos que hay 7877 imágenes del número cero y 6903 imágenes del número cero. Por último, determinamos si el conjunto de datos está o no balanceado. Para esto elegimos un umbral de tolerancia igual a 0,1, luego miramos el módulo de la resta de proporciones de cada clase, y si el módulo es menor al umbral puesto. Ya que la proporción obtenida es menor a 0,1 consideramos que el dataframe está balanceado.
b) Separamos los datos en conjuntos de entrenamiento (train) y prueba (test), utilizando una proporción del 80% para entrenamiento y 20% para prueba.

c) Utilizamos el algoritmo K-Nearest Neighbors (KNN) para entrenar y evaluar a los modelos de clasificación binaria. Probamos diferentes combinaciones de tres y cuatro atributos (píxeles), elegidos en base de un análisis exploratorio previo.

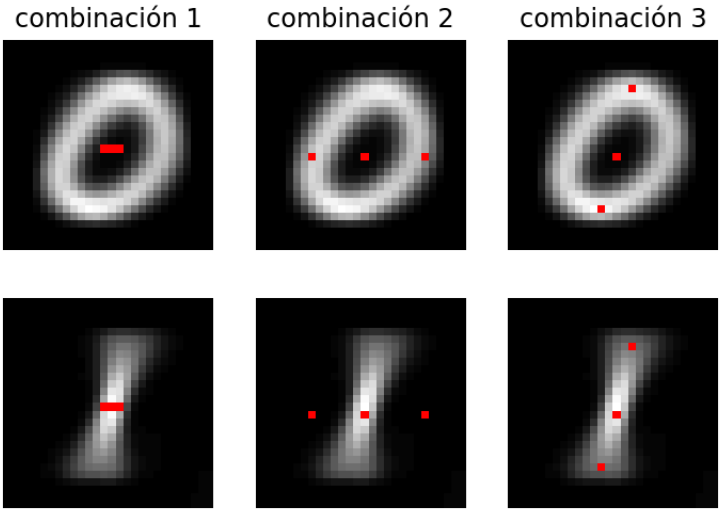

Figura XII. Representación de las imágenes promedio generadas marcando los píxeles utilizados en cada combinación.

La primera combinación da una exactitud de 0.8836 de exactitud. El segundo conjunto da una exactitud de 0.9882 y el tercero una exactitud de 0.9581.

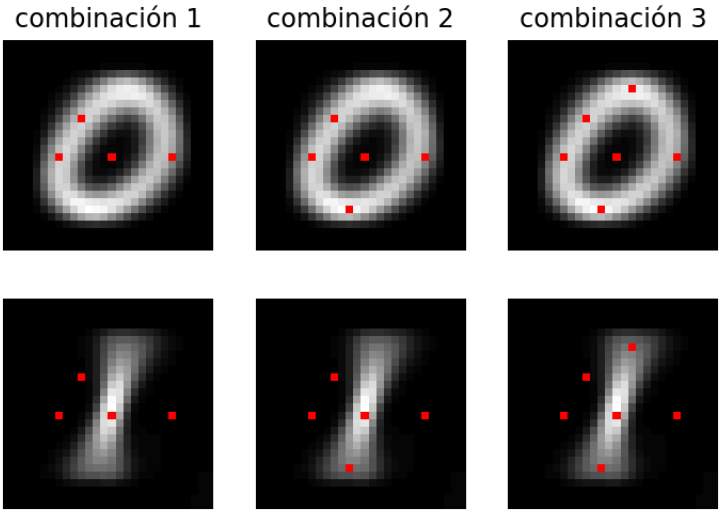

Figura XIII. Representación de las imágenes promedio generadas marcando los píxeles utilizados en cada combinación.

La primera combinación que utilizamos es 4 atributos. Cuando calculamos la exactitud nos da un 0.9885 de exactitud. Luego probamos con un conjunto de 5 atributos, cuya exactitud nos da un 0.9871 de exactitud. Por último probamos con un conjunto de 6 atributos, cuya exactitud nos da un 0.9868 de exactitud.

d) Comparamos el accuracy de los modelos con distintos valores de k (número de vecinos) según cada conjunto

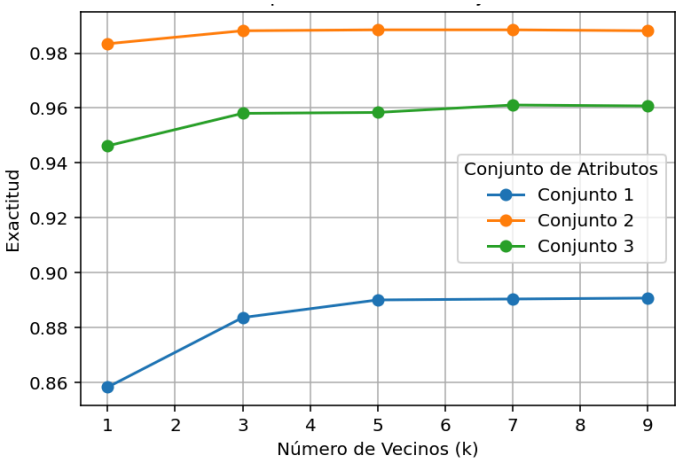

Figura XIV. Gráfico del promedio de exactitud en función del número de vecinos (k) diferenciado por conjuntos

Podemos ver como en el conjunto 2 logramos el modelo con mejor exactitud, mientras que el primer modelo es el que tiene menor accuracy.

Ejercicio 3: Clasificación multiclase

El objetivo de este punto es predecir dada una imagen, a cual de los 10 dígitos corresponde. El proceso incluye la preparación de los datos, la selección de alturas del árbol, el entrenamiento del modelo, la evaluación de su rendimiento utilizando métricas de clasificación y el análisis de sus hiper parámetros.

a) Separamos el conjunto de desarrollo en *x_dev, y_dev* tomando el 80% de los datos. Por otro lado, tenemos el conjunto de validación *x_heldout*, *y_heldout*, que corresponde al 20% de los datos. Nos aseguramos que la cantidad de imágenes de cada dígito están relativamente balanceadas entre sí (no hay grandes diferencias entre la cantidad de imágenes que tiene cada número). Igualmente utilizamos *stratify* para procurar que la proporción de clases sea la misma en ambos conjuntos.

b y c) Ajustamos un modelo de árbol de decisión probando con profundidades del 1 al 10. También realizamos una validación cruzada con 5 folds. Almacenamos la precisión (*accuracy*) para cada fold y profundidad, obteniendo como resultado el siguiente gráfico.

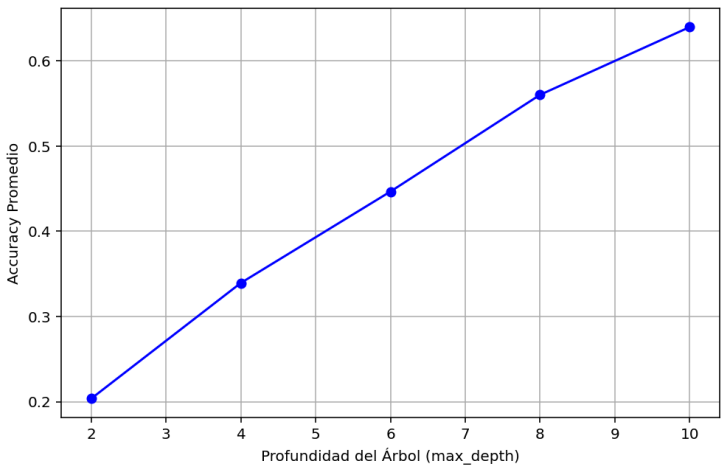

Figura XIV. Gráfico del promedio de exactitud en función de las profundidades del árbol

Podemos ver como aumenta la *accuracy* a medida que aumenta la profundidad del árbol, obteniendo como mejor profundidad el 10. Además, usamos *GridSearchCV* para buscar la mejor combinación de hiperparámetros (*max_depth, min_samples_split, min_samples_leaf, criterion -utilizamos gini, que miden la impureza del modelo-)* con validación cruzada estratificada.

Estos hiper parámetros controlan cómo se construye y se comporta el árbol, y ajustarlos correctamente es necesario para obtener un modelo que generalice bien. De esta forma, evitamos el sobreajuste, haciendo que el árbol no sea demasiado complejo y pueda generalizar bien nuevos datos. Por otro lado, equilibramos el sesgo y la varianza, porque un árbol simple puede tener alto sesgo y no capturar patrones importantes. Mientras que un árbol complejo tiene alta varianza y es muy sensible a los datos de entrenamiento.

Debido a lo costoso que es correr un árbol de decisión con un dataset tan grande, optamos por quedarnos con un nivel de accuracy regular.

d) Entrenamos el mejor modelo en todo el conjunto de desarrollo *X_dev, y_dev.* Luego obtenemos la precisión en held-out, con los hiper parámetros óptimos. Obtuvimos como mejor modelo encontrado: *DecisionTreeClassifier (max_depth=9, min_samples_leaf = 2,min_samples_split = 5, random_state = 42). Nuestros hiperparámetros óptimos: {'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 2, 'min_samples_split': 5}*. La accuracy en validación cruzada: 0.6083, y por último, la precisión en el conjunto de validación (held-out): 0.62. El reporte de clasificación tiene la precisión, el recall y el f1-score, con su respectiva profundidad. Graficamos nuestra matriz de confusión, que muestra cuántas predicciones fueron correctas e incorrectas para cada clase.

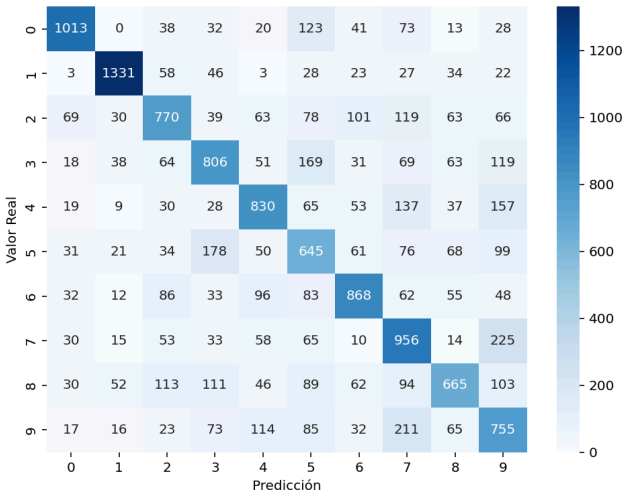

Figura XV. Matriz de confusión de nuestro modelo de árbol de decisión.

Como comentamos en el análisis exploratorio, podemos ver como los números que son más similares entre sí tienden a ser los que más se confunden.

Conclusión

Con los valores obtenidos concluimos en que KNN es el modelo que mejor se ajusta para predecir dos dígitos con este dataset. Esto es porque se basa en los k-neighbors más cercanos, además de que usamos un subconjunto de imágenes mucho más acotado. Decidir si un número es un 0 o un 1 no es semejante a intentar predecir qué dígito es una imagen entre 10 valores diferentes, más teniendo en cuenta la magnitud del dataset. El Decision Tree se transformó en un modelo muy complejo, lo que nos resultó muy costoso de procesar. Además, fue muy difícil encontrar hiper parámetros que maximicen el accuracy en este caso, por lo que no conseguimos un buen modelo.