Machine Learning
Proyecto 2

Autores
Willian Berrocal Alvarado
John Rodrigo Monroy Angeles
Tania Araceli Barreda Galvez
Bianca Brunella Aguinaga Pizarro
Melisa Karen Rivera Alagon
Luis Robledo

Docente
Rensso Victor Hugo Mora Colque

Fecha
2 de octubre del 2024

1. Introducción

En este proyecto, se busca implementar un sistema de clasificación para el conjunto de datos MNIST, el cual contiene imágenes de dígitos escritos a mano. El objetivo principal es explorar diferentes métodos de extracción de características y algoritmos de clasificación, evaluando el desempeño de cada uno de ellos mediante métricas estándar como precisión, recall y F1-score. Para ello, se utilizará la librería Torch para manejar el dataset y se implementarán los siguientes métodos:

- Extracción de características mediante wavelets o algún otro método que permita generar vectores característicos por cada imagen.
- Clasificación utilizando regresión logística, extendida para n clases.
- Clasificación utilizando Máquinas de Vectores de Soporte (SVM), adaptadas para múltiples clases.
- Clasificación utilizando el algoritmo K-Nearest Neighbors (KNN).
- Clasificación utilizando árboles de decisión.
- Generación de gráficas de la función de pérdida vs. época para aquellos algoritmos que involucren la optimización de parámetros mediante funciones de pérdida.

Se realizarán experimentos dividiendo el conjunto de datos en tres subconjuntos: entrenamiento (70 %), validación (15 %) y prueba (15 %). Además, se empleará una validación cruzada mediante k-fold cross-validation o bootstrap para mejorar la robustez de los resultados.

Finalmente, se generará una tabla con las métricas de evaluación y los hiperparámetros de cada modelo, permitiendo comparar el desempeño de los algoritmos implementados y determinar cuál de ellos ofrece los mejores resultados para el conjunto de datos en cuestión.

Enlace al código: Repositorio del proyecto

2. Generación de Vectores Característicos

Para la generación de vectores característicos se usó "Haar Wavelet", una técnica matemática que pertenece a la familia de las wavelets y es utilizada para transformar una señal (en nuestro caso, una imagen) en una representación que resuma tanto la frecuencia como la localización de los patrones en la señal [1].

La transformada Haar descompone la imagen en varios niveles de detalle, generando coeficientes que representan la imagen de diferentes maneras:

- **Coeficiente de aproximación:** Contiene la versión más simplificada de la imagen (como una vista de baja resolución o borrosa).
- **Coeficientes de detalle:** Estos representan los detalles de la imagen en términos de bordes y diferencias entre los píxeles. Se dividen en tres componentes:
  - *Horizontal (cH):* Los bordes horizontales en la imagen.
  - *Vertical (cV):* Los bordes verticales.
  - *Diagonal (cD):* Los bordes en direcciones diagonales.

En resumen, Haar wavelet proporciona una serie de coeficientes que pueden ser tratados como el **vector característico** de la imagen. Estos coeficientes capturan la información más importante en una forma comprimida, que luego puede ser utilizada por un modelo de aprendizaje automático.

Por ejemplo, si se tuviese la imagen de un dígito "5" en MNIST, la transformada Haar proporcionaría una versión "simplificada" del 5 (el coeficiente de aproximación), junto con las líneas importantes que forman los bordes del número (los coeficientes de detalle), características que son más fáciles de procesar por los algoritmos de clasificación.

3. Clasificación por Regresión Logística

La regresión logística es una técnica para predecir la pertenencia de una muestra a una clase binaria. Cuando el problema implica más de dods clases, como en el dataset MNIST, en donde hay 10 clases, se emplea la regresión logística multinomial.

La regresión logística multinomial extiende la regresión logística binaria para manejar problemas de clasificación con múltiples clases. En vez de predecir una clase binaria, predice probabilidades para K clases diferentes. Para ello, se emplea la función softmax, que generaliza la función sigmoide [2], empleada en la regresión logística binaria.

3.1. Función Softmax

La función softmax transforma las salidas de la función en probabilidades para cada una de las clases. Genera una distribución de probabilidades que suma 1. Emplea la siguiente fórmula:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Donde:

- z_i es la salida lineal para la clase i
- K es el número de clases

3.2. Función de Pérdida: Entropía Cruzada Categórica

La entropía cruzada categórica mide la discrepancia entre las probabilidades predichas por el modelo y las etiquetas verdaderas [2] y se emplea como función de pérdida en problemas multiclase.

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \log(\hat{y}_{ij})$$

Donde:

- N es el número de ejemplos,
- K es el número de clases,
- y_{ij} es la etiqueta verdadera para la muestra i y la clase j,
- $\hat{y}_{ij}$ es la probabilidad predicha por el modelo para la clase j.

3.3. Función de optimización: Gradiente descendiente

Gradiente descendiente es un algoritmo de optimización empleado para minimizar la función de pérdida. En cada iteración, los pesos y los sesgos del modelo son ajustados en la dirección opuesta al gradiente de la función de pérdida con respecto a esos parámetros en cada iteración [3]. Permite al modelo mejorar gradualmente sus predicciones al ajustar los pesos para minimizar la función de pérdida.

$$w = w - \alpha \nabla L(w)$$

Donde:

- $\nabla L(w)$ es el gradiente de la función de pérdida con respecto a los pesos w,
- α es la tasa de aprendizaje, un hiperparámetro que controla el tamaño de los pasos en la dirección del gradiente.

4. Clasificación por SVM

El algoritmo SVM se usa principalmente para problemas de clasificación binaria. Sin embargo, para este proyecto, extenderemos el enfoque a múltiples clases. Los métodos más comunes son:

- Uno vs uno: k Se entrena un clasificador svm para cada par de clases. Si hay n clases, se necesitarán n(n-1)/2 clasificadores. Cada clasificador decide entre dos clases, y la clase con más votos gana.
- Uno vs todos: k Se entrena un clasificador SVM para cada clase contra todas las demás. Para n clases, se entrena n clasificadores. Cada clasificador predice si una instancia pertenece a su clase o no. La clase con la puntuación más alta es la elegida.

4.1. Fórmulas utilizadas

4.1.1. Hiperplano Separador

La ecuación del hiperplano separador en un espacio d-dimensional se expresa como:

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{1}$$

- $\mathbf{w}$: Vector de pesos que determina la orientación del hiperplano.
- $\mathbf{x}$: Vector de características de un punto de datos.
- b: Término de sesgo (bias) que desplaza el hiperplano.

4.1.2. Margen

El margen es la distancia entre el hiperplano y los puntos de datos más cercanos (vectores de soporte). Para los puntos de soporte, se tiene:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i \tag{2}$$

- y_i: Clase del punto i (1 para la clase positiva, -1 para la negativa).
- $\mathbf{x}_i$: Vector de características del punto i.

4.1.3. Optimización del Margen

La tarea de SVM es maximizar el margen, que se puede formalizar como minimizar la siguiente función objetivo:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{3}$$

sujeta a:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i \tag{4}$$

- $\|\mathbf{w}\|^2$: Norma cuadrada del vector de pesos, que indica el "tamaño"del hiperplano.

4.1.4. Función de Decisión para N Clases (One-vs-All)

Para un problema de clasificación multiclase, la decisión se toma mediante la siguiente regla para el clasificador k:

$$f_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k \tag{5}$$

La clase predicha se determina como:

$$\hat{y} = \arg\max_k f_k(\mathbf{x}) \tag{6}$$

4.1.5. Función de Pérdida

La función de pérdida utilizada en SVM es la pérdida hinge, que se define como:

$$L(\mathbf{w}, b) = \sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \tag{7}$$

- N: Número total de muestras.
- Esta función penaliza a los puntos que están en el lado incorrecto del margen.

4.1.6. Problema de Optimización Regularizado

Para incluir la regularización, la función objetivo se convierte en:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \tag{8}$$

- C: Hiperparámetro que controla el trade-off entre maximizar el margen y minimizar la clasificación errónea.

4.1.7. Resumen

- **Hiperplano separador**: Define la frontera entre clases.
- **Margen**: Medida de separación entre las clases.
- **Optimización**: Minimiza la norma de los pesos para maximizar el margen.
- **Función de decisión**: Para clasificar en múltiples clases mediante One-vs-All.
- **Pérdida hinge**: Penaliza errores de clasificación.
- **Regularización**: Controla el trade-off entre complejidad y ajuste.

5. Clasificación por KNN

El algoritmo K-Nearest Neighbors (K-NN) es un método supervisado de Machine Learning que se utiliza tanto para tareas de clasificación como de regresión [2]. Funciona identificando los K puntos más cercanos a una nueva muestra basándose en la distancia, lo que le permite hacer predicciones sobre esa muestra [4]. En clasificación, asigna la clase más común entre los vecinos cercanos, mientras que en regresión estima un valor continuo a partir de estos puntos similares aprendidos durante la etapa de entrenamiento [2]. En resumen, el objetivo principal es encontrar los vecinos más cercanos a un punto de consulta dado para asignarle una etiqueta de clase o predecir su valor [5].

5.1. Métricas de distancia

El algoritmo necesita definir métricas de distancia, que permiten calcular la cercanía entre el punto de consulta y los demás puntos de datos [5][6]. Estas distancias son fundamentales para establecer límites de decisión que dividen el espacio en regiones, determinando a qué clase o valor pertenece el nuevo punto [5][6].

Existen varios métodos para medir la distancia entre puntos, como la distancia euclidiana, la distancia Manhattan, la distancia de Minkowski y la distancia de Hamming [5][6].

5.2. Distancia Euclidiana

En este caso, se usará la Distancia Euclidiana, que mide la distancia directa entre dos puntos en un espacio multidimensional [5][6].

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

5.3. Definición del k

El valor de k en el algoritmo define cuántos vecinos serán considerados para determinar la clasificación del punto de consulta específico [5][6]. Por ejemplo, sí $k = 1$, la instancia se asignará a la misma clase que su vecino más cercano. Sí $k = 2$, se considerarán los dos vecinos más cercanos, y la instancia se asignará a la clase más frecuente entre ellos.

- Valores bajos de k pueden hacer que el modelo sea más sensible a las pequeñas variaciones en los datos de entrenamiento, lo que puede resultar en una alta varianza, pero bajo sesgo [5][6].
- Valores más altos de k suavizan la decisión del algoritmo, ya que toman en cuenta más vecinos. Esto disminuye la varianza y aumenta el sesgo, ya que el modelo se vuelve menos flexible y puede no captar bien la complejidad de los datos [5][6].

5.4. Selección del valor óptimo de k

Lo ideal es encontrar un equilibrio entre variación y sesgo [5][6]. A menudo, un buen punto de partida es usar la raíz cuadrada del número total de instancias en el conjunto de datos de entrenamiento como valor de k [6]. Es recomendable un valor impar para k para evitar empates en la clasificación, y la estrategia de validación cruzada puede ser útil para hallar el valor óptimo [5][6]. Por otro lado, es importante también tener en cuenta la distribución de los datos. Si los datos contienen ruido o muchos valores atípicos, es recomendable optar por un valor de k más alto, ya que esto permite suavizar la influencia de los puntos atípicos [5].

6. Clasificación por Árboles de Decisión

El árbol de decisión se construye de manera recursiva, dividiendo los datos en función de las características que mejor separan las clases. Se define una clase `Node` que representa cada nodo del árbol. Cada nodo es una decisión basada en una característica específica y un umbral, o una hoja que asigna una clase. A continuación, se describen los componentes clave de esta implementación:

6.1. Cálculo de la Entropía

La entropía mide la impureza de un conjunto de datos. Por ejemplo, si tenemos un conjunto de datos en el que todas las respuestas son "sí"pues la entropía es cero. Esto se utiliza para determinar qué tan desordenados están los datos en términos de las clases. Y se define con la siguiente fórmula [7] :

$$H(Y) = -\sum_{i=1}^{C} p_i \log_2 p_i \tag{9}$$

6.2. Ganancia de la información

La ganancia de información evalúa la efectividad de una característica para dividir los datos en clases puras. Se calcula como la diferencia entre la entropía antes de la división y la entropía promedio después de la división.

$$IG(Y, X) = H(Y) - \left(\frac{|Y_{\text{left}}|}{|Y|} H(Y_{\text{left}}) + \frac{|Y_{\text{right}}|}{|Y|} H(Y_{\text{right}}) \right) \tag{10}$$

6.3. Construcción Recursiva del Árbol

El árbol se construye de manera recursiva, dividiendo los datos en función de las características que mejor separan las clases. Este proceso continúa hasta que se cumplen ciertos **criterios de parada**, que evitan que el árbol crezca indefinidamente y ayudan a prevenir el sobreajuste. A continuación, se detallan los criterios de parada considerados en el código:
Criterios de Parada

- **Profundidad Máxima (`max_depth`):** Este criterio limita la profundidad del árbol, es decir, el número máximo de niveles que el árbol puede tener desde la raíz hasta las hojas. En la implementación proporcionada, se decidió una profundidad máxima de 10 niveles.
- **Número Mínimo de Muestras para Dividir (`min_samples_split`):** Este criterio especifica el número mínimo de muestras requeridas para intentar dividir un nodo. En este caso, se establece en 2. Si un nodo contiene menos muestras que este umbral, no se realiza una división adicional, y el nodo se convierte en una hoja que asigna una clase.
- **Pureza de las Clases:** Si todas las muestras en un nodo pertenecen a la misma clase, el nodo se considera puro y no se realizan más divisiones. Esta condición asegura que el árbol no continúe dividiéndose innecesariamente cuando ya se ha alcanzado una separación completa de las clases.

Solo si ninguno de estos criterios se cumple, el algoritmo busca la mejor división posible del nodo actual basándose en la ganancia de información. Este enfoque garantiza que el árbol se construya de manera eficiente, equilibrando la complejidad del modelo con su capacidad de generalización.

6.4. Predicción y Evaluación

Una vez construido el árbol, se utiliza para predecir las clases de nuevas muestras. La predicción se realiza recorriendo el árbol desde la raíz hasta una hoja, siguiendo las decisiones basadas en las características.

En cuanto a la evaluación, se han utilizado métricas como la precisión, el informe de clasificación y la matriz de confusión. Estas métricas proporcionan una visión completa de cómo el modelo se desempeña en la clasificación de diferentes clases [7] y permiten identificar posibles áreas de mejora, como podría ser en caso la matriz de confusión de muchos falsos positivos o la precisión sea muy baja.

6.5. Visualización de resultados

Para la visualización del árbol obtenido se usó la librería graphix, de esta manera se facilita su visualización gráfica, que muestra las decisiones tomadas en cada nodo y cómo se ramifican las clases, la imagen obtenida se guarda en formato pdf, ya que, al ser un árbol muy grande el resultado en consola no se aprecia de la manera correcta. Asimismo, se seleccionó solo una muestra de 1000 imágenes del conjunto de entrenamiento para entrenar el árbol de decisión. Esta elección se fundamenta principalmente debido a la eficiencia computacional y visualización de los resultados.

7. Experimentos

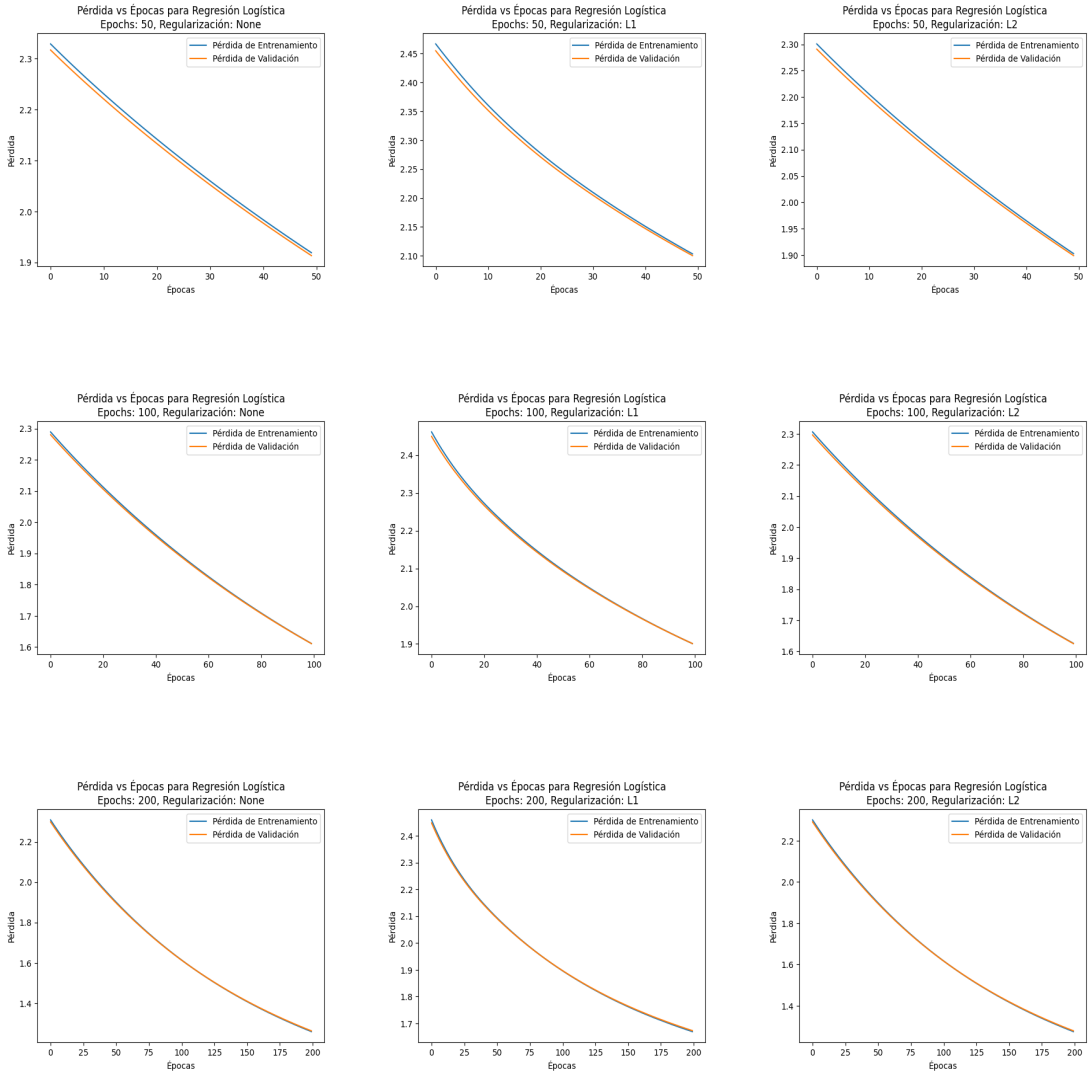
7.1. Regresión Logística

Para la experimentación con regresión logística se tuvo en cuenta el uso de las regularizaciones L1 y L2, para evaluar en caso de un posible sobreajuste.

7.1.1. Configuración experimental

Para cada experimento se empleó el método de entrenamiento K-Fold Cross Validation con una tasa de aprendizaje de 0.01. Se tomaron en cuenta 50, 100 y 200 épocas. Para las regularizaciones, se usó como parámetro lambda a 0.01.

7.1.2. Resultados



7.1.3. Observaciones

- La regularización L2 muestra un rendimiento más estable en las métricas, en comparación a las otras implementaciones. Este resultado es coherente con lo esperado de este modelo,

num_epochs	regularization	precision	recall	f1_score
50	L1	0.738413	0.6803	0.650219
50	L2	0.727201	0.6700	0.642316
100	L1	0.747996	0.7111	0.681036
100	L2	0.760384	0.7298	0.708101
200	L1	0.770395	0.7514	0.736320
200	L2	0.782143	0.7685	0.761232

Cuadro 1: Resultados de precisión, recall y F1-Score con diferentes épocas y regularizaciones

ya que esta regularización minimiza los coeficientes sin llegar a que alguno sea cero, reduciendo el sobreajuste.

- La regularización L1 tiende a mostrar un rendimiento menor en precisión, aunque exhibe un recall ligeramente superior en las primeras iteraciones. Esto puede deberse a que reduce a cero algunos coeficientes, simplificando el modelo, lo que no sería muy efectivo para este caso.
- La precisión y el F1-Score tienden a estabilizarse en las 200 épocas, por lo que este valor podría ser adecuado para ajustar el modelo.
- El modelo con regularización L2 muestra equilibrio entre precisión y recall. Si bien es cierto que el modelo sin regularizafción tiende a tener un valor de F1-Score mayor, presenta mayor variación de recall, lo que puede indicar un posible sobreajuste.
- A partir de las gráficas, si bien se halla que existe una convergencia suave en las tres implementaciones, la regularización L2 provee más consistente y pareja a mayor cantidad de épocas.

7.1.4. Modelo final

Tras haber realizado las pruebas de hiperparámetros, podríamos tomar el modelo de regresión logística multinomial con regularización L2 con 200 épocas, puesto que parece ser la opción más estable en capacidad de generalización, manteniendo un balance entre recall y precisión. Sin embargo, de verificarse la cantidad de épocas, podría emplearse sin regularización, puesto que también presenta un ajuste generalizado para este conjunto de datos.

7.2. SVM

Para la experimentación con SVM, se utilizó la validación cruzada K-Fold para evaluar el modelo con diferentes valores de lambda, con el objetivo de observar el rendimiento en términos de precisión, recall y F1-score.

7.2.1. Configuración experimental

Los experimentos se llevaron a cabo con tres valores de lambda: 0.001, 0.01 y 0.1. Se empleó una tasa de aprendizaje de 0.001 y un total de 500 iteraciones. La evaluación de rendimiento se realizó sobre un conjunto de datos de prueba.

7.2.2. Resultados

7.2.3. Observaciones

- Con un valor de lambda de 0.001, el modelo SVM logró una precisión del 70.80 % y un F1-score de 0.7066, lo que indica un rendimiento aceptable en la clasificación.

Valor de Lambda	Precisión Promedio	Recall Promedio	F1-Score Promedio
0.001	0.7080	0.7010	0.7066
0.01	0.6070	0.6019	0.6193
0.1	0.0960	0.1082	0.0317

Cuadro 2: Resultados del modelo SVM para diferentes valores de lambda.

- La disminución en el rendimiento con un valor de lambda de 0.01, con una precisión de 60.70 % y un F1-score de 0.6193, sugiere que este valor puede estar llevando a un sobreajuste al modelo.
- El uso de un lambda de 0.1 resultó en un rendimiento deficiente, con una precisión de solo 9.60 % y un F1-score de 0.0317, indicando que el modelo no está generalizando bien y probablemente está subajustado.
- Se observa que un menor valor de lambda puede mejorar la capacidad de generalización del modelo, a medida que la penalización de los parámetros se reduce.
- El tiempo de entrenamiento aumentó considerablemente con valores de lambda más altos, lo que sugiere una mayor complejidad en el modelo.
- A partir de los resultados, se recomienda utilizar el valor de lambda de 0.001 para lograr un equilibrio adecuado entre precisión y recall en el modelo SVM.

7.2.4. Modelo final

Con base en los experimentos realizados, se sugiere adoptar el modelo SVM con un valor de lambda de 0.001, ya que proporciona un rendimiento equilibrado en términos de precisión, recall y F1-score, manteniendo una buena capacidad de generalización sobre el conjunto de datos.

7.3. Árbol de Decisión

En esta sección, se presenta el rendimiento del modelo de Árbol de Decisión utilizando diferentes profundidades máximas.

7.3.1. Configuración experimental

Para la experimentación, se evaluaron tres configuraciones diferentes del modelo de Árbol de Decisión, variando el parámetro `max_depth` en 5, 10 y 15. Se utilizó el método K-Fold Cross Validation para la evaluación del rendimiento.

7.3.2. Resultados

Max Depth	Precisión	Recall	F1 Score
5	0.6150	0.6100	0.6100
10	0.6250	0.6200	0.6200
15	0.6250	0.6200	0.6200

Cuadro 3: Resultados del Árbol de Decisión para diferentes profundidades máximas

7.3.3. Observaciones

- El modelo de Árbol de Decisión con `max_depth` de 5 presenta un rendimiento moderado, con una precisión del 61.50 % y un F1 Score de 0.61. Esto indica que el modelo tiene una capacidad limitada para generalizar los datos.
- Aumentar la profundidad máxima a 10 resulta en una ligera mejora en la precisión y en las métricas de recall y F1 Score, alcanzando valores del 62.50 %. Sin embargo, este rendimiento se estabiliza al aumentar a 15, donde las métricas permanecen constantes.
- La variación en el rendimiento entre los diferentes valores de `max_depth` es mínima, lo que sugiere que, dentro de este rango, el modelo no se beneficia significativamente de una mayor complejidad.
- Los resultados indican que el modelo tiene un desempeño relativamente similar para profundidades de 10 y 15, sugiriendo que el aumento en la complejidad del modelo no resulta en una mejora considerable del rendimiento.

7.3.4. Modelo final

Considerando los resultados obtenidos, se sugiere utilizar el modelo de Árbol de Decisión con `max_depth` de 10, ya que proporciona un equilibrio entre precisión y complejidad. Sin embargo, dada la similaridad en el rendimiento, se podría optar por `max_depth` de 5 como una opción más simple y eficiente.

7.4. KNN

Para la experimentación con KNN, se evaluó el modelo con diferentes valores de k para observar el rendimiento en términos de precisión, recall y F1-score.

7.4.1. Configuración experimental

Los experimentos se realizaron con tres valores de k: 3, 5 y 7. Se utilizó un conjunto de datos de prueba y se compararon las métricas de rendimiento entre las diferentes configuraciones de k.

7.4.2. Resultados

Valor de k	Precisión Promedio	Recall Promedio	F1-Score Promedio
3	0.9763	0.9761	0.9761
5	0.9755	0.9753	0.9754
7	0.9747	0.9745	0.9746

Cuadro 4: Resultados del modelo KNN para diferentes valores de k.

7.4.3. Observaciones

- Con un valor de k de 3, el modelo KNN logró una precisión del 97.63 % y un F1-score de 0.9761, lo que muestra un rendimiento excelente en la clasificación.
- Al aumentar el valor de k a 5, se observa una pequeña disminución en las métricas, con una precisión de 97.55 % y un F1-score de 0.9754. Esto sugiere que el modelo sigue funcionando bien, pero con un ligero ajuste en las predicciones debido a la consideración de más vecinos.

- Con un valor de k de 7, las métricas bajan un poco más, con una precisión del 97.47 % y un F1-score de 0.9746, lo que indica que a medida que aumenta k, se suavizan las predicciones, pero el rendimiento sigue siendo alto.
- Se observa que, en este caso, el valor óptimo de k está en el rango de 3 a 5, donde se logra el mejor equilibrio entre precisión, recall y F1-score.

7.4.4. Modelo final

A partir de los resultados experimentales, se recomienda utilizar un valor de k igual a 3, ya que proporciona las métricas de rendimiento más altas en términos de precisión y F1-score. Sin embargo, el modelo con k igual a 5 también ofrece un rendimiento competitivo con un mayor margen de suavización en las predicciones.

8. Conclusiones

8.1. Regresión Logística

- **L1 vs. L2**: La regularización L2 mostró un mejor rendimiento en términos de precisión, *recall* y *F1-Score*, especialmente con 200 épocas. L1 tiende a tener menor precisión y una menor estabilidad, aunque su *recall* fue ligeramente superior al inicio.
- **Mejor configuración**: El modelo con regularización L2 y 200 épocas resultó ser el más equilibrado, con un *F1-Score* de 0.761232, lo que indica una buena capacidad de generalización y menor riesgo de sobreajuste.
- **Observaciones adicionales**: A mayor cantidad de épocas, el modelo tiende a estabilizarse. Es posible utilizar el modelo sin regularización si se verifica una convergencia adecuada con un número mayor de épocas.

8.2. SVM

- **Lambda 0.001**: Este valor mostró el mejor rendimiento, con una precisión promedio de 70.80 % y un *F1-Score* de 0.7066, lo que indica que el modelo puede clasificar bien los datos.
- **Lambda 0.01 y 0.1**: Los valores más altos de lambda disminuyeron significativamente el rendimiento del modelo, con un subajuste severo para lambda de 0.1.
- **Mejor configuración**: Se recomienda utilizar un valor de lambda de 0.001, ya que proporciona un mejor balance entre precisión y *recall* sin caer en sobreajuste.

8.3. Árbol de Decisión

- **Max Depth 5 vs. 10 vs. 15**: No se observaron mejoras significativas al aumentar la profundidad del árbol. La precisión y el *F1-Score* se estabilizaron en 62.50 % para profundidades de 10 y 15, lo que sugiere que el aumento de complejidad no aporta grandes beneficios.
- **Mejor configuración**: Se recomienda una profundidad máxima de 10 para un buen balance entre rendimiento y complejidad, aunque se podría optar por 5 para simplificar el modelo sin una gran pérdida de precisión.

8.4. KNN

- **k=3, 5, 7**: A medida que aumenta el valor de k, las métricas de rendimiento disminuyen ligeramente. Sin embargo, el modelo sigue siendo muy preciso con valores de k pequeños.
- **Mejor configuración**: Se recomienda utilizar k=3, ya que logró la mayor precisión (97.63 %) y el mejor *F1-Score* (0.9761), proporcionando las mejores predicciones.

9. Bibliografía

Referencias

[1] W. contributors. "Haar wavelet." (jul. de 2024), dirección: https://en.wikipedia.org/wiki/Haar_wavelet.

[2] Na. "Algoritmo K-Nearest Neighbor — Aprende Machine Learning." (jul. de 2020), dirección: https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python.

[3] IBM. "¿Qué es el Descenso de Gradiente?" (2024), dirección: https://www.ibm.com/mx-es/topics/gradient-descent.

[4] R. Díaz. "Algoritmo KNN – Cómo funciona y ejemplos en Python." (jul. de 2024), dirección: https://www.themachinelearners.com/algoritmo-knn/#%C2%BFComo_funciona_el_modelo_KNN.

[5] IBM. "KNN: ¿Qué es KNN?" [Accessed: Oct. 2, 2024]. (mayo de 2023), dirección: https://www.ibm.com/mx-es/topics/knn.

[6] "¿Qué es k vecino más cercano (kNN)? — Una guía integral sobre k vecino más cercano." [Accessed: Oct. 2, 2024]. (), dirección: https://www.elastic.co/es/what-is/knn#how-does-knn-work.

[7] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, n.º 1, págs. 81-106, 1986. dirección: https://doi.org/10.1007/BF00116251.