Clasificador MLP

Alumno: Fabian Neftaly Guia Cruz

Fecha: 15 de diciembre de 2025

Analisis Exploratorio de Datos (EDA)

El dataset contiene `1,688` registros y `17` variables, de las cuales `8` son numéricas y `9` categóricas, siendo col17 la variable objetivo categórica con `7` clases.

No existen valores faltantes en ninguna columna, lo que simplifica el análisis posterior y descarta la necesidad inmediata de imputación. La variable objetivo se encuentra razonablemente balanceada, sin una clase dominante extrema.

Columnas numericas: 8

'col2'	'col3'	'col4'	'col7'	'col8'	'col11'	'col13'	'col14'

Columnas categoricas: 9

'col1'	'col5'	'col6'	'col9'	'col10'	'col12'	'col15'	'col16'	'col17'

Las variables numéricas presentan alta cardinalidad (muchos valores únicos), lo que sugiere que son continuas o cuasi-continuas.

En términos de distribución, la mayoría son aproximadamente simétricas, aunque destacan algunos casos:

- `col2` muestra una cola positiva marcada (sesgo > 1) con presencia relevante de outliers, mientras que `col8` presenta una cola negativa fuerte y una gran cantidad de valores extremos según el criterio IQR.
- `col14` exhibe inflación de ceros (Q1 = 0), lo que podría indicar un fenómeno discreto o una variable activada solo bajo ciertas condiciones.

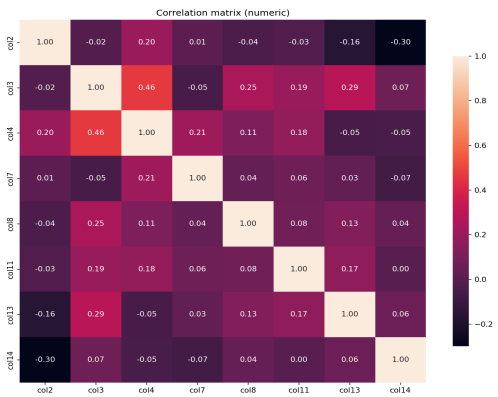
En general, los outliers están concentrados principalmente en `col2` y `col8`, mientras que el resto de las variables numéricas son relativamente estables.

En cuanto a correlaciones, la mayoría de las relaciones lineales entre variables numéricas son débiles, lo que sugiere baja redundancia. La única relación moderada observable es entre `col3` y `col4 (r ≈ 0.46)`, lo que podría indicar información parcialmente compartida entre ambas. No se observan correlaciones fuertes que anticipen problemas de multicolinealidad severa.

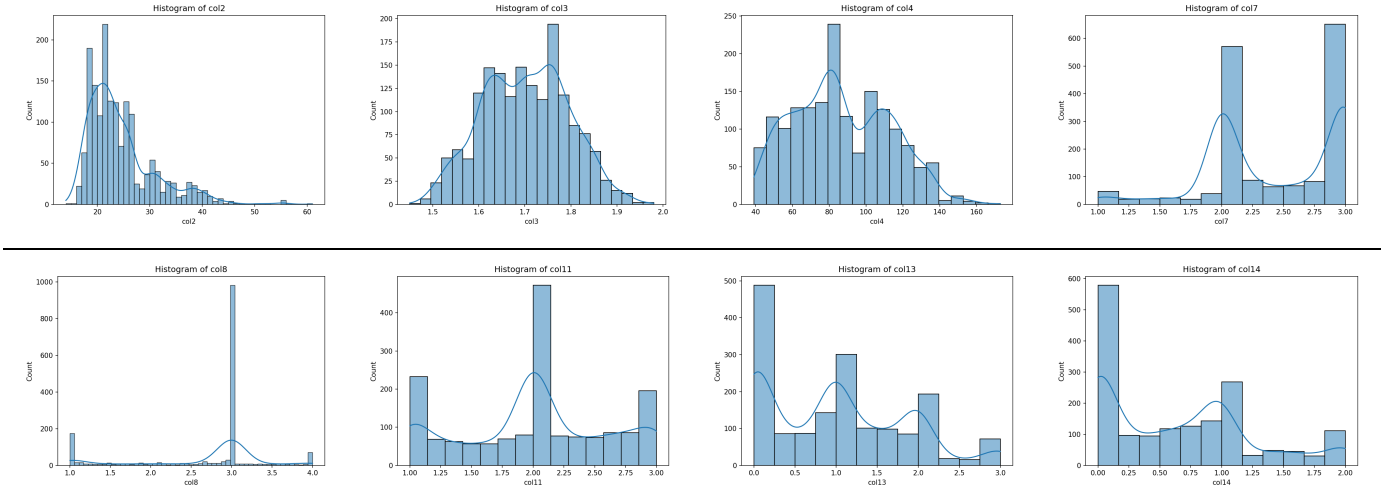
Las variables categóricas muestran un patrón claro de desbalance interno:

> En la mayoría de ellas, una sola categoría concentra más del 80% de las observaciones (por ejemplo, col5, col6, col9, col10 y col12).
>
> Esto indica baja entropía informativa en varias columnas categóricas, aunque siguen siendo potencialmente útiles en combinación con otras variables.
>
> Es importante notar que, aunque compartan etiquetas como cat_1 o cat_2, las categorías no son comparables entre columnas, ya que son independientes por definición.

Mapa de calor de correlaciones



Histogramas de variables numéricas



Justificacion y Desiciones

Antes del entrenamiento del modelo se aplicó un proceso de preprocesamiento con el objetivo de transformar las variables originales a un formato numérico adecuado para una red neuronal multicapa (MLP), garantizando estabilidad numérica, convergencia eficiente y evitando sesgos indebidos entre características.

El preprocesamiento se implementó mediante `Pipeline` y `ColumnTransformer` de *scikit-learn*, lo que asegura reproducibilidad, consistencia entre las fases de entrenamiento, validación y prueba, y previene fugas de información (*data leakage*), ya que todas las transformaciones se ajustan exclusivamente con los datos de entrenamiento.

Para las variables numéricas se incluyó una imputación por mediana, elegida por su robustez frente a valores atípicos, aunque en este conjunto de datos no se detectaron valores faltantes. Posteriormente, se aplicó una transformación de potencia Yeo-Johnson para aproximar las distribuciones a la normalidad, estabilizar la varianza y reducir asimetrías, incluso en presencia de valores cero o negativos. Finalmente, se utilizó un escalado estándar (`StandardScaler`) para centrar las variables en media cero y desviación estándar uno, lo cual es fundamental para que todas las características contribuyan de forma equilibrada al proceso de optimización de la red neuronal.

En el caso de las variables categóricas, se utilizó imputación por la moda, aunque tampoco se identificaron valores faltantes. Estas variables se transformaron mediante one-hot encoding, una técnica adecuada para variables nominales sin orden inherente, evitando la introducción de relaciones artificiales entre categorías. Se empleó además la opción `handle_unknown='ignore'` para garantizar un comportamiento robusto ante categorías no vistas durante la inferencia.

Finalmente, tanto el preprocesador como el codificador de etiquetas fueron persistidos usando `joblib`. Esto permite reutilizar exactamente las mismas transformaciones en validación, prueba y despliegue, asegurando coherencia en las predicciones y evitando la necesidad de recalcular estadísticas del conjunto de datos.

Arquitectura del modelo MLP

Capa de entrada

La dimensión de entrada del modelo se define dinámicamente como:

```
input_dim = X_train.shape[1]
```

Esto corresponde al número total de características después del preprocesamiento, incluyendo:

- Variables numéricas escaladas, Variables categóricas codificadas mediante One-Hot Encoding

Esta estrategia garantiza que el modelo sea agnóstico al número exacto de columnas generadas por el preprocesamiento.

Capas ocultas

El modelo utiliza dos capas ocultas densas:

```
```

```
hidden_layers = (64, 32)
```

Primera capa oculta – 64 neuronas

- Función principal: capturar interacciones complejas entre variables de entrada. Tamaño suficientemente grande para aprender relaciones no lineales. Adecuada para un dataset de tamaño medio como el utilizado.

Segunda capa oculta – 32 neuronas

- Reduce progresivamente la dimensionalidad. Obliga al modelo a concentrar la información relevante. Actúa como regularización estructural.

Función de activación

En todas las capas ocultas se utilizó:

```
activation = 'relu'
```

Justificación de ReLU

- Introduce no linealidad. Evita el problema del vanishing gradient. Es computacionalmente eficiente. Funciona especialmente bien con datos escalados

ReLU es una elección estándar y adecuada para redes neuronales profundas y problemas de clasificación.

Regularización

Para reducir el riesgo de sobreajuste, se incorporaron dos técnicas de regularización: **Dropout y Regularización L2**

```
dropout_rate = 0.15, l2_reg = 5e-5
```

- Desactiva aleatoriamente el 15% de las neuronas durante el entrenamiento. Evita la dependencia excesiva de neuronas específicas. Mejora la capacidad de generalización.
- Penaliza pesos excesivamente grandes. Favorece soluciones más suaves y estables. Complementa el uso de Dropout.

Compilación del modelo

El modelo se compila con:

- Función de pérdida:

  > sparse_categorical_crossentropy
  > (adecuada para etiquetas enteras codificadas)

- Optimizador:

  > Adam con tasa de aprendizaje 1e-3, Adaptativo, Convergencia rápida

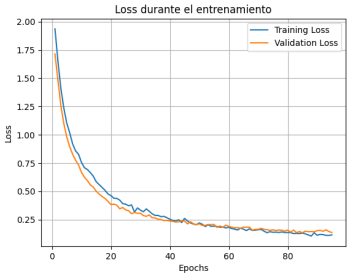
Estable para este tipo de arquitectura

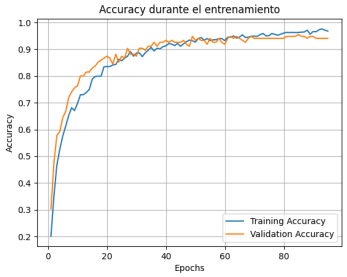
- Métrica:

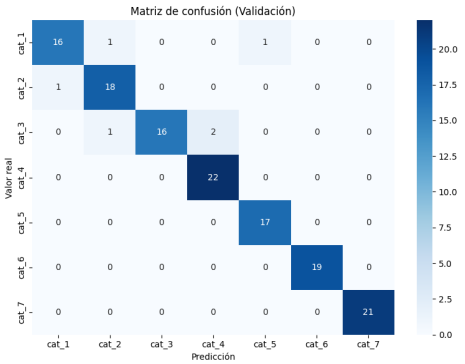
  > accuracy

Proceso de entrenamiento, validación y test

Validación



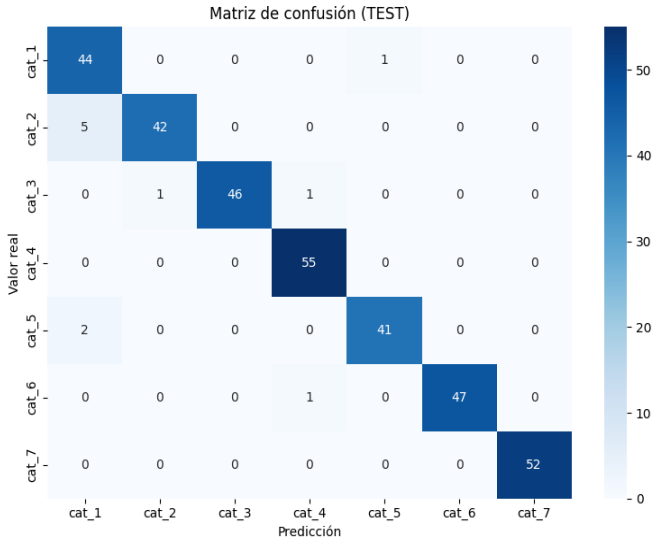




Reporte de clasificación (VALIDACIÓN):

	precision	recall	f1-score	support
cat_1	0.94	0.95	0.92	19
cat_3	1.00	0.84	0.91	19
cat_4	0.92	1.00	0.96	22
cat_5	0.94	1.00	0.97	17
cat_6	1.00	1.00	1.00	19
cat_7	1.00	1.00	1.00	21
-	-	-	-	-
accuracy	-	-	0.96	135
macro avg	0.96	0.95	0.95	135
weighted avg	0.96	0.96	0.95	135

Test



Reporte de clasificación (TEST):

	precision	recall	f1-score	support
cat_1	0.86	0.98	0.92	45
cat_2	0.98	0.89	0.93	47
cat_3	1.00	0.96	0.98	48
cat_4	0.96	1.00	0.98	55

	precision	recall	f1-score	support
cat_5	0.98	0.95	0.96	43
cat_6	1.00	0.98	0.99	48
cat_7	1.00	1.00	1.00	52
-	-	-	-	-
accuracy	-	-	0.97	338
macro avg	0.97	0.97	0.97	338
weighted avg	0.97	0.97	0.97	338

En el conjunto de validación, el modelo alcanzó una exactitud del 96%, con valores altos y equilibrados de precision, recall y f1-score tanto en los promedios macro como ponderados. Esto indica que el modelo logra un desempeño consistente entre todas las clases, sin favorecer de manera significativa a las categorías con mayor número de muestras. La mayoría de las clases presentan valores de f1-score superiores a 0.90, y varias de ellas alcanzan un desempeño perfecto, lo que sugiere que la arquitectura y el preprocesamiento permiten una correcta separación entre categorías. Al evaluar el modelo en el conjunto de prueba, se observa una ligera mejora en el rendimiento global, alcanzando una exactitud del 97%. Los promedios macro y ponderado de precision, recall y f1-score se mantienen alineados y cercanos a 0.97, lo que confirma que el modelo generaliza adecuadamente y no presenta signos de sobreajuste. La estabilidad de las métricas entre validación y prueba refuerza la idea de que el modelo aprendió patrones representativos del problema y no dependientes del conjunto de entrenamiento.

Como ejecutar

1. Abrir el proyecto:
    - Descomprimir el archivo ZIP descargado. Navegar a la carpeta del proyecto. Abrir una terminal en esa ubicación.
2. Crear un entorno virtual (opcional pero recomendado):

```
python -m venv venv
source venv/bin/activate  # En Windows: venv\Scripts\activate
```

3. Instalar dependencias:

```
pip install -r requirements.txt
```

4. Ejecutar la aplicación:

```
python src/main.py
# siempre ejecutar desde la raiz del proyecto para evitar errores de rutas, no ejecutar
desde la carpeta src.
```

5. El modelo preentrenado esta en la carpeta `src/test/mlp_model.keras`, el preprocesador en `src/test/preprocessor.joblib` y el codificador de etiquetas en `src/test/label_encoder.joblib`. Estos archivos son necesarios para realizar predicciones en nuevos datos de test.
6. Para ejecutar el modelo preentrenado en un nuevo conjunto de datos, colocar el archivo CSV en la carpeta `src/test/` y seleccionar la opción de predicción en el menú principal. Asegurarse de que el archivo tenga el mismo formato y columnas que el conjunto de datos original utilizado para el entrenamiento.
7. Seguir las instrucciones en pantalla para completar las acciones de predicción.
8. Al terminar se mostrara un reporte de clasificación en la consola y una matriz de confusión con los resultados obtenidos en `src/test/confusion_matrix_test.png`.