Registro Diario de Avances – Modelo basado en IA para detección de somnolencia

**Fecha:** 21/05/2025

**Autor: Tirza Buendia**

**Versión:** 1.0

# 1. Introducción

Este documento detalla el seguimiento diario del Sprint 1 del proyecto modelo basado en IA para detección de somnolencia, centrado en el desarrollo de funcionalidades básicas. Se incluyen actividades diarias, tareas completadas, evidencias de código, un burndown chart y un análisis del rendimiento del equipo.

# 2. Lineamientos de Diseño

Desarrollar una primera versión funcional del modelo basado en IA para detección de somnolencia en conductores que permita a los usuarios:

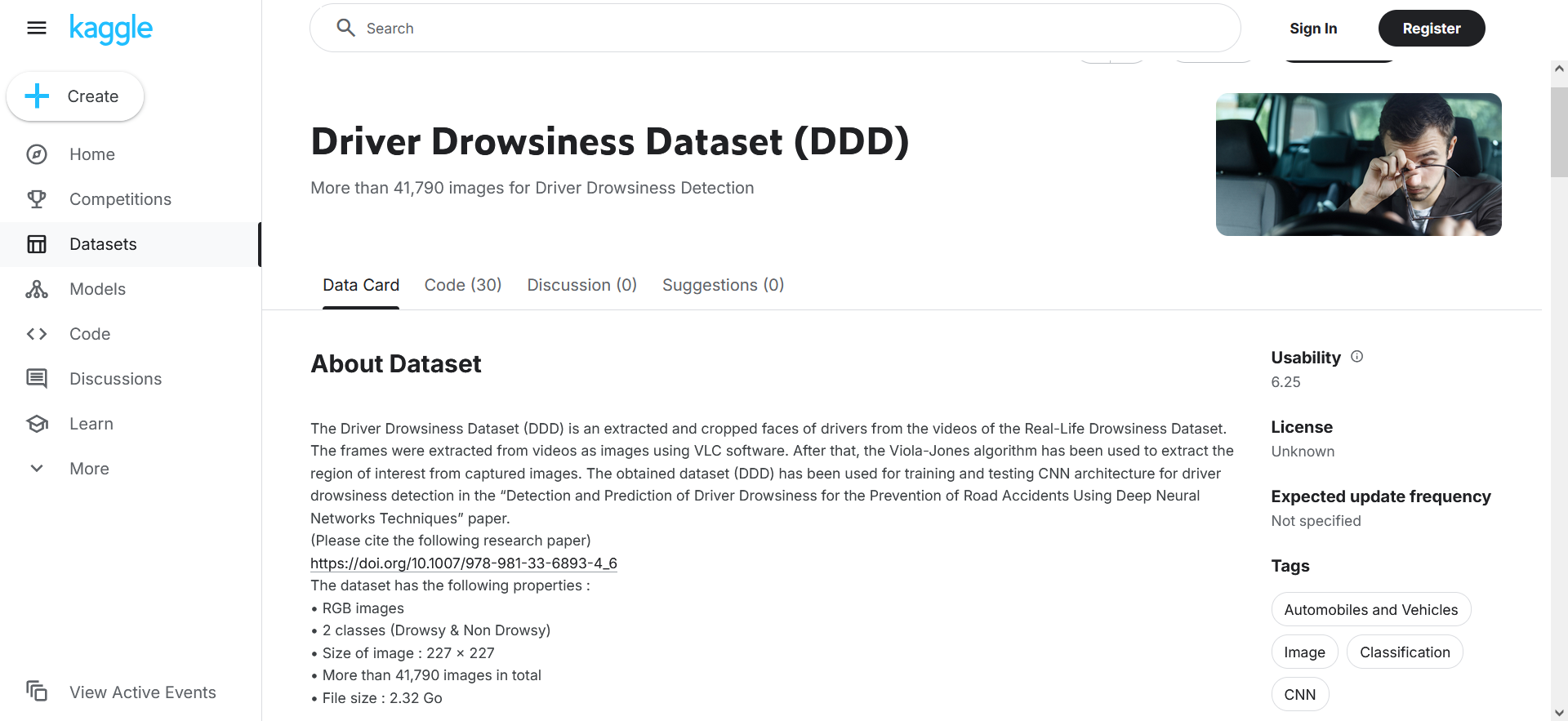
* Usar detección facial en el modelo.
* Monitorear el cierre ocular.

# 3. Historias de Usuario y Tareas

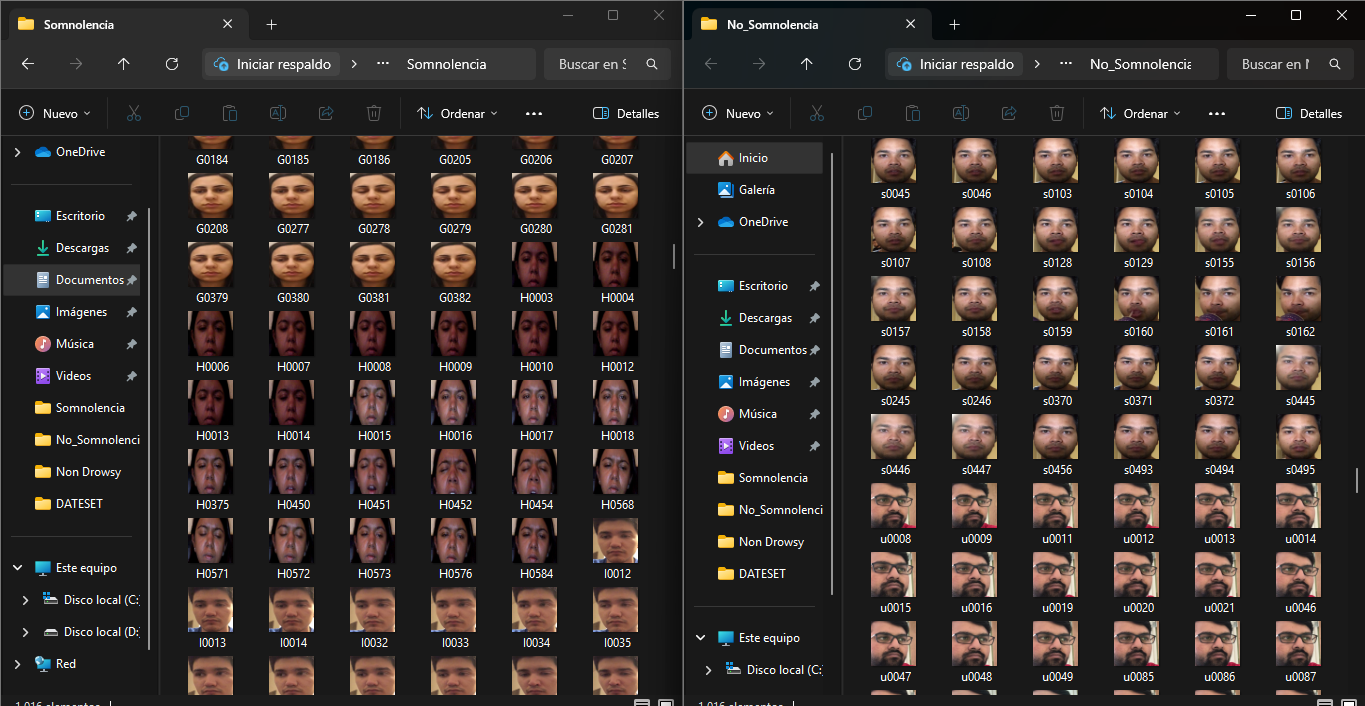
| Backlog ID | Historia de Usuario | Tiempo Estimado | Fecha de Inicio | Fecha de Finalización |
| --- | --- | --- | --- | --- |
| HU-0.1 | Recolección y Etiquetado del dataset | 9 | 08/05/2025 | 10/05/2025 |
| HU-0.2 | Preprocesamiento de datos | 6 | 11/05/2025 | 14/05/2025 |
| HU-0.3 | Entrenamiento del Modelo | 8 | 15/05 | 20/05/2025 |

## 3.1 Historia de Usuario 0.2 – Recolección y etiquetado del dataset

* **Actividad 0.1.1 Recolectar Frames:** se usó una dataset pública de Kaggle, Driver Drowsiness Dataset (DDD). <https://www.kaggle.com/datasets/ismailnasri20/driver-drowsiness-dataset-ddd>



* **Actividad 0.1.2 Etiquetar Frames:** Se separó un grupo de 1013 frames para la clase somnolencia y 1013 frames para la clase de no somnolencia a partir de la dataset original que cuenta con +42000 frames.



## 3.2 Historia de Usuario 0.3 – Entrenamiento del modelo

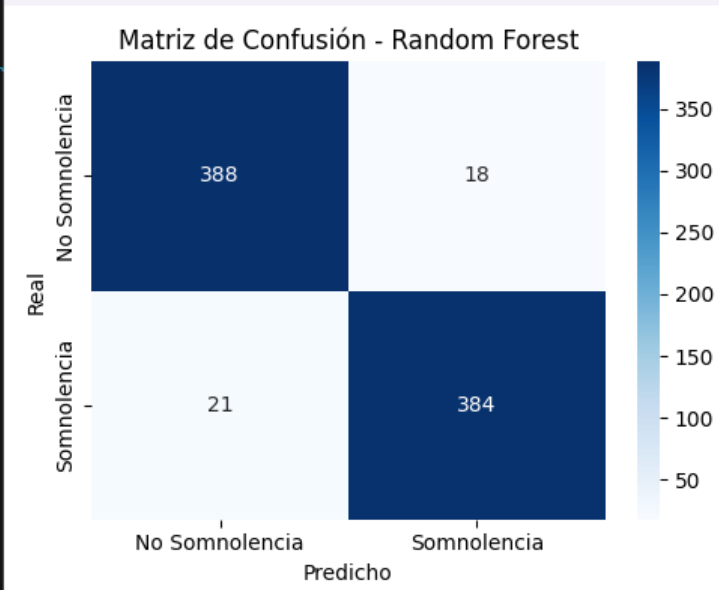
* **Actividad 0.3.1 Seleccionar algoritmo y definir features:** Se definieron los features en base a las métricas recolectadas en los frames del dataset y la selección del algoritmo se hizo con ayuda del análisis de un cuadro comparativo.
  + Features: EAR (Eye aspect ratio), MAR (Mouth aspect ratio), Ángulos de Tait-Bryan (Pitch, Yaw, Roll).
  + Potenciales algoritmos a usar: Se decidió tomar 2 algoritmos (RF y XGBoost debido a su buen rendimiento bajo las consideraciones del tipo de datos y hardware con el que se cuenta para el proyecto) (documento completo: <https://docs.google.com/document/d/13DHo6f-SFOGolDbunmPCI4tlgLuc9-XXutow92xk-nI/edit?usp=sharing>)

| **Característica** | **Random Forest (RF)** | **SVM** | **XGBoost** | **KNN** | **Regresión Logística** |
| --- | --- | --- | --- | --- | --- |
| **Facilidad de aplicación** | Media | Media | Baja | Alta | Alta |
| **Datos no lineales** | **Media** | **Alta (con kernel)** | **Alta** | **Baja** | **Baja** |
| **Tolerancia a datos ruidosos** | Alta | Baja | Media | Baja | Media |
| **Esfuerzo computacional** | Medio | Alto | Alto | Alto | Bajo |
| **Grandes medianas** | Medio | Bajo | Alta | Bajo | Alta |
| **Tareas de clasificación** | Sí | Sí | Sí | Sí | Sí |
| **Requiere datos estandarizados** | No | Sí | No | Sí | Sí |
| **Escalabilidad** | Media | Baja | Alta | Baja | Alta |
| **Riesgo de sobreajuste** | Bajo | Bajo | Medio-alto (en datasets pequeños) | Alto | Bajo |

* **Actividad 0.3.4 Evaluar con validación cruzada + test final:** Para la evaluación de ambos modelos se usó una metodología híbrida (nested cross validation), mezcla de k-fold con train-test split (80/20). Se pudo analizar la evaluación según las métricas de rendimiento establecidas .

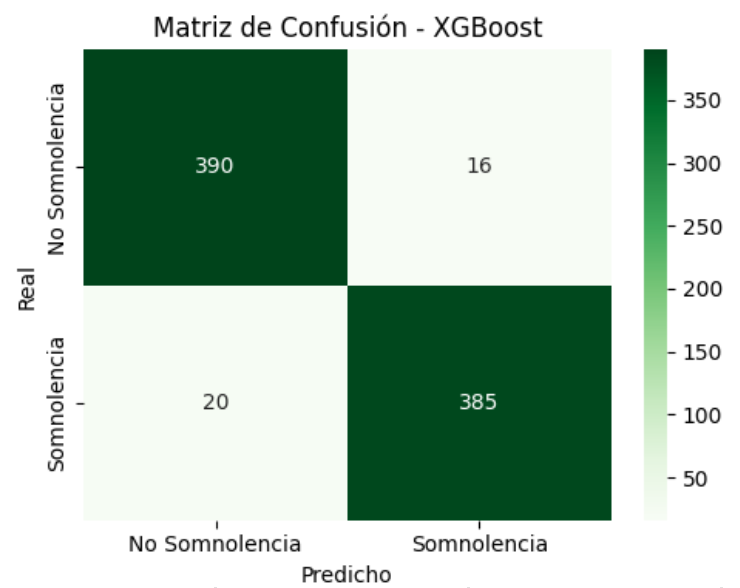
| **RF - Entrenamiento con validación cruzada** | | | |
| --- | --- | --- | --- |
| **Métrica** | **No Somnolencia** | **Somnolencia** | **Accuracy** |
| Precision | 0.93 | 0.95 | 0.94 |
| Recall | 0.95 | 0.93 |  |
| F1-Score | 0.94 | 0.94 |  |
| Support | 1620 | 1621 | 3241 |

| **RF - Test Final** | | | |
| --- | --- | --- | --- |
| **Métrica** | **No Somnolencia** | **Somnolencia** | **Accuracy** |
| Precision | 0.95 | 0.96 | 0.95 |
| Recall | 0.96 | 0.95 |  |
| F1-Score | 0.95 | 0.95 |  |
| Support | 406 | 405 | 811 |



| **XGB - Entrenamiento con validación cruzada** | | | |
| --- | --- | --- | --- |
| **Métrica** | **No Somnolencia** | **Somnolencia** | **Accuracy** |
| Precision | 0.93 | 0.94 | 0.94 |
| Recall | 0.94 | 0.93 |  |
| F1-Score | 0.94 | 0.93 |  |
| Support | 1620 | 1621 | 3241 |

| **XGB - TEST FINAL** | | | |
| --- | --- | --- | --- |
| **Métrica** | **No Somnolencia** | **Somnolencia** | **Accuracy** |
| Precision | 0.95 | 0.96 | 0.96 |
| Recall | 0.96 | 0.95 |  |
| F1-Score | 0.96 | 0.96 |  |
| Support | 406 | 405 | 811 |



| **Comparativa de test final** | | | |
| --- | --- | --- | --- |
| **Métrica** | **Clase** | **Random Forest (RF)** | **XGBoost (XGB)** |
| Accuracy | Global | 0.95 | **0.96** |
| Precision | No Somnolencia | 0.95 | 0.95 |
| Precision | Somnolencia | 0.96 | 0.96 |
| Recall | No Somnolencia | 0.96 | 0.96 |
| Recall | Somnolencia | 0.95 | 0.95 |
| F1-Score | No Somnolencia | 0.95 | **0.96** |
| F1-Score | Somnolencia | 0.95 | **0.96** |

En base a los resultados obtenidos, ambos algoritmos muestran un buen desempeño en la tarea de clasificación. Sin embargo, al analizar las métricas del conjunto de prueba (compuesto por datos no vistos durante el entrenamiento) se observa que el algoritmo XGBoost presenta una ligera ventaja sobre Random Forest. En las matrices de confusión, XGBoost reduce tanto los falsos positivos como los falsos negativos, lo cual es crítico en la detección de somnolencia. Además, en el test final, ambos modelos obtuvieron un Recall del 96% y una Precision del 95%, pero XGBoost alcanzó un F1-Score de 96% y Accuracy (predicciones acertadas) del 96%, superando al modelo RF por 1%. Si bien la validación cruzada fue fundamental para comparar modelos de forma objetiva y prevenir sobreajuste, la selección final se basó en el rendimiento con datos completamente nuevos. Por estas razones, **se eligió XGBoost como el modelo definitivo para la implementación**.

# 4. Evidencias de Código

## 4.1 Historia de Usuario 0.2 — Preprocesamiento de datos

Archivo: Modelo/extraccion\_metricas.py

import cv2

import mediapipe as mp

import pandas as pd

import os

import math

import numpy as np

# Función para calcular distancia euclidiana

def euclidean(p1, p2):

return ((p1[0]-p2[0])\*\*2 + (p1[1]-p2[1])\*\*2)\*\*0.5

# Función para calcular EAR (ambos ojos)

def compute\_ear(landmarks):

# Ojo izquierdo

left = [landmarks[i] for i in [33, 160, 158, 133, 153, 144]]

left\_ear = (euclidean(left[1], left[5]) + euclidean(left[2], left[4])) / (2.0 \* euclidean(left[0], left[3]))

# Ojo derecho

right = [landmarks[i] for i in [362, 385, 387, 263, 373, 380]]

right\_ear = (euclidean(right[1], right[5]) + euclidean(right[2], right[4])) / (2.0 \* euclidean(right[0], right[3]))

return (left\_ear + right\_ear) / 2.0

# Función para calcular MAR

def compute\_mar(landmarks):

mouth = [landmarks[i] for i in [61, 291, 81, 178, 13, 14]]

mar = (euclidean(mouth[2], mouth[3]) + euclidean(mouth[4], mouth[5])) / (2.0 \* euclidean(mouth[0], mouth[1]))

return mar

# Función para calcular ángulos de orientación de la cabeza

def compute\_tait\_bryan\_angles(landmarks\_3d, img\_shape):

ref\_indices = [1, 152, 33, 263, 61, 291] # nariz, barbilla, ojos, boca

image\_points = np.array([(landmarks\_3d[i].x \* img\_shape[1], landmarks\_3d[i].y \* img\_shape[0]) for i in ref\_indices], dtype="double")

model\_points = np.array([

(0.0, 0.0, 0.0), # nariz

(0.0, -63.0, -12.0), # barbilla

(-43.0, 32.0, -26.0), # ojo izq

(43.0, 32.0, -26.0), # ojo der

(-28.0, -28.0, -24.0), # boca izq

(28.0, -28.0, -24.0) # boca der

])

focal\_length = img\_shape[1]

center = (img\_shape[1] / 2, img\_shape[0] / 2)

camera\_matrix = np.array([[focal\_length, 0, center[0]],

[0, focal\_length, center[1]],

[0, 0, 1]], dtype="double")

dist\_coeffs = np.zeros((4, 1)) # Sin distorsión

success, rotation\_vector, \_ = cv2.solvePnP(model\_points, image\_points, camera\_matrix, dist\_coeffs)

if not success:

return None, None, None

rotation\_mat, \_ = cv2.Rodrigues(rotation\_vector)

sy = math.sqrt(rotation\_mat[0, 0] \*\* 2 + rotation\_mat[1, 0] \*\* 2)

singular = sy < 1e-6

if not singular:

pitch = math.degrees(math.atan2(rotation\_mat[2, 1], rotation\_mat[2, 2]))

yaw = math.degrees(math.atan2(-rotation\_mat[2, 0], sy))

roll = math.degrees(math.atan2(rotation\_mat[1, 0], rotation\_mat[0, 0]))

else:

pitch = math.degrees(math.atan2(-rotation\_mat[1, 2], rotation\_mat[1, 1]))

yaw = math.degrees(math.atan2(-rotation\_mat[2, 0], sy))

roll = 0

return pitch, yaw, roll

# Configurar FaceMesh

mp\_face\_mesh = mp.solutions.face\_mesh

face\_mesh = mp\_face\_mesh.FaceMesh(static\_image\_mode=True)

# Función para procesar imágenes de un directorio

def procesar\_frames(directorio, clase):

datos = []

for filename in os.listdir(directorio):

if filename.lower().endswith(('.jpg', '.png')):

ruta = os.path.join(directorio, filename)

img = cv2.imread(ruta)

if img is None:

continue

rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

results = face\_mesh.process(rgb)

if results.multi\_face\_landmarks:

for face in results.multi\_face\_landmarks:

h, w, \_ = img.shape

puntos = [(int(p.x \* w), int(p.y \* h)) for p in face.landmark]

ear = compute\_ear(puntos)

mar = compute\_mar(puntos)

pitch, yaw, roll = compute\_tait\_bryan\_angles(face.landmark, img.shape)

if pitch is None:

continue

datos.append([filename, ear, mar, pitch, yaw, roll, clase])

return datos

# Directorios de entrada y salida

ruta\_somnolencia = r"C:\ruta\a\frames\_sin\_metricas\somnolencia"

ruta\_no\_somnolencia = r"C:\ruta\a\frames\_sin\_metricas\no\_somnolencia"

ruta\_salida = r"C:\ruta\a\frames\_sin\_metricas\metricas\_somnolenciaf.csv"

# Procesamiento y guardado

datos\_somnolencia = procesar\_frames(ruta\_somnolencia, 1)

datos\_no\_somnolencia = procesar\_frames(ruta\_no\_somnolencia, 0)

df = pd.DataFrame(datos\_somnolencia + datos\_no\_somnolencia,

columns=["frame", "EAR", "MAR", "Pitch", "Yaw", "Roll", "Clase"])

df.to\_csv(ruta\_salida, index=False)

print("¡CSV guardado correctamente en:", ruta\_salida)

Archivo: Modelo/escalado\_caracteristicas.py

#sin rutas absolutas, cuando los archivos se manejan dentro de la carpeta del programa

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import pandas as pd

try:

df = pd.read\_csv("metricas\_somnolencia.csv")

# Columnas por tipo

minmax\_cols = ['EAR', 'MAR']

standard\_cols = ['Pitch', 'Yaw', 'Roll']

# Aplicar escalado

df[minmax\_cols] = MinMaxScaler().fit\_transform(df[minmax\_cols])

df[standard\_cols] = StandardScaler().fit\_transform(df[standard\_cols])

# Exportar si deseas probar con el modelo

df.to\_csv("datos\_escalados\_mixto.csv", index=False)

except Exception as e:

print(f"Error durante el procesamiento: {str(e)}")

Archivo: Modelo/aumentacion\_datos.py

import cv2

import os

import random

import numpy as np

# --- Configuración ---

INPUT\_BASE = r"C:\Users\USER\Documents\DATESET\DS\_filtrada" #Usa doble barra o raw string (r"...") es la dirección de donde salen las frames

OUTPUT\_BASE = r"C:\Users\USER\Documents\DATESET\DS\_filtrada\frames\_aumentadass" #es la dirección donde se guardan las frames

os.makedirs(OUTPUT\_BASE, exist\_ok=True) #Crea la carpeta si no existe

# --- Funciones de aumento ---

# Ajuste de brillo y contraste (TU COMENTARIO ORIGINAL)

def ajustar\_brillo(img):

"""Brillo aleatorio suave (-20 a +20)""" #Rango de brillo más realista y seguro

brillo = random.randint(-30, 30)

return cv2.convertScaleAbs(img, beta=brillo)

# Rotación de imagen (TU COMENTARIO ORIGINAL)

def rotar\_imagen(img, angulo):

"""Rotación con bordes reflejados (para no perder información)""" #BORDER\_REFLECT evita artefactos

(h, w) = img.shape[:2]

centro = (w // 2, h // 2)

M = cv2.getRotationMatrix2D(centro, angulo, 1.0)

return cv2.warpAffine(img, M, (w, h), borderMode=cv2.BORDER\_REFLECT)

# Aumentar con aleatoriedad controlada (TU COMENTARIO ORIGINAL + MEJORA)

def aplicar\_aumento(img, clase):

"""Aplicar 1 transformación aleatoria, controlada por clase"""

# Opciones ponderadas (50% brillo, 50% rotación) # weights asegura balance

opcion = random.choices(['brillo', 'rotacion'], weights=[0.5, 0.5], k=1)[0]

if opcion == 'brillo':

return ajustar\_brillo(img)

else:

# Rotación más amplia para somnolencia (-15° a +15° vs ±5°)

angulo = random.uniform(-15, 15) if clase == "somnolencia" else random.uniform(-5, 5)

return rotar\_imagen(img, angulo)

# --- Procesamiento por clase --- (TU COMENTARIO ORIGINAL + MANEJO DE ERRORES)

def aumentar\_clase(clase):

input\_dir = os.path.join(INPUT\_BASE, clase)

output\_dir = os.path.join(OUTPUT\_BASE, clase)

os.makedirs(output\_dir, exist\_ok=True) #Asegura que exista la subcarpeta

total\_ok = 0

total\_error = 0

for filename in os.listdir(input\_dir):

if filename.endswith((".jpg", ".png")): #Soporta ambos formatos

try:

path = os.path.join(input\_dir, filename)

img = cv2.imread(path)

if img is None:

raise ValueError(f"Error al leer {filename}")

# Aplicar aumento y guardar (TU LÓGICA ORIGINAL)

img\_aug = aplicar\_aumento(img, clase)

nombre\_salida = f"aug\_{filename}"

cv2.imwrite(os.path.join(output\_dir, nombre\_salida), img\_aug)

total\_ok += 1

except Exception as e:

print(f"⚠️ Error en {filename}: {str(e)}")

total\_error += 1

continue

# Mensaje detallado por clase

print(f"\nResultados para clase '{clase}':")

print(f" - Frames procesados: {total\_ok}")

print(f" - Errores: {total\_error}")

return total\_ok > 0 # Retorna True si al menos 1 frame se procesó

# --- Ejecución --- (CON REPORTE FINAL MEJORADO)

if \_\_name\_\_ == "\_\_main\_\_":

print(" Iniciando data augmentation...\n")

# Procesar ambas clases

resultado\_somnolencia = aumentar\_clase("somnolencia")

resultado\_no\_somnolencia = aumentar\_clase("no\_somnolencia")

# ✨ Reporte consolidado

if resultado\_somnolencia and resultado\_no\_somnolencia:

print("\n✅ ¡Proceso completado con éxito!")

print(f" - Frames aumentados guardados en: {OUTPUT\_BASE}")

else:

print("\n❌ ¡Proceso completado con errores!")

print(" Verifica los mensajes anteriores.")

## 4.1 Historia de Usuario 0.3 — Entrenamiento del modelo

Archivo: Modelo/modelo\_rf.py

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_predict, StratifiedKFold

from sklearn.metrics import classification\_report, confusion\_matrix

# Cargar train y test

train = pd.read\_csv("train.csv")

test = pd.read\_csv("test.csv")

X\_train = train[["EAR", "MAR", "Pitch", "Yaw", "Roll"]]

y\_train = train["Clase"]

X\_test = test[["EAR", "MAR", "Pitch", "Yaw", "Roll"]]

y\_test = test["Clase"]

# Configurar validación cruzada

kfold = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

# Crear el modelo

modelo\_rf = RandomForestClassifier(random\_state=42)

# Validación cruzada con los datos de entrenamiento

y\_pred\_cv = cross\_val\_predict(modelo\_rf, X\_train, y\_train, cv=kfold)

print("\n=== Random Forest - Validación Cruzada ===")

print(classification\_report(y\_train, y\_pred\_cv, target\_names=["No Somnolencia", "Somnolencia"]))

# Entrenamiento final

modelo\_rf.fit(X\_train, y\_train)

# Evaluación final en test

y\_pred\_test = modelo\_rf.predict(X\_test)

print("\n=== Random Forest - Test Final ===")

print(classification\_report(y\_test, y\_pred\_test, target\_names=["No Somnolencia", "Somnolencia"]))

# Matriz de confusión

cm = confusion\_matrix(y\_test, y\_pred\_test)

plt.figure(figsize=(5, 4))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Somnolencia", "Somnolencia"], yticklabels=["No Somnolencia", "Somnolencia"])

plt.title("Matriz de Confusión - Random Forest")

plt.xlabel("Predicho")

plt.ylabel("Real")

plt.tight\_layout()

plt.show()

Archivo: Modelo/modelo\_xgb.py

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from xgboost import XGBClassifier

from sklearn.model\_selection import cross\_val\_predict, StratifiedKFold

from sklearn.metrics import classification\_report, confusion\_matrix

# Cargar train y test

train = pd.read\_csv("train.csv")

test = pd.read\_csv("test.csv")

X\_train = train[["EAR", "MAR", "Pitch", "Yaw", "Roll"]]

y\_train = train["Clase"]

X\_test = test[["EAR", "MAR", "Pitch", "Yaw", "Roll"]]

y\_test = test["Clase"]

# Configurar validación cruzada

kfold = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

# Crear el modelo

modelo\_xgb = XGBClassifier(random\_state=42, use\_label\_encoder=False, eval\_metric='logloss')

# Validación cruzada

y\_pred\_cv = cross\_val\_predict(modelo\_xgb, X\_train, y\_train, cv=kfold)

print("\n=== XGBoost - Validación Cruzada ===")

print(classification\_report(y\_train, y\_pred\_cv, target\_names=["No Somnolencia", "Somnolencia"]))

# Entrenamiento final

modelo\_xgb.fit(X\_train, y\_train)

# Evaluación final en test

y\_pred\_test = modelo\_xgb.predict(X\_test)

print("\n=== XGBoost - Test Final ===")

print(classification\_report(y\_test, y\_pred\_test, target\_names=["No Somnolencia", "Somnolencia"]))

# Matriz de confusión

cm = confusion\_matrix(y\_test, y\_pred\_test)

plt.figure(figsize=(5, 4))

sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", xticklabels=["No Somnolencia", "Somnolencia"], yticklabels=["No Somnolencia", "Somnolencia"])

plt.title("Matriz de Confusión - XGBoost")

plt.xlabel("Predicho")

plt.ylabel("Real")

plt.tight\_layout()

plt.show()

Archivo: Modelo/limpieza\_split\_datos.py

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Cargar el CSV original

df = pd.read\_csv("TOTALmetricas\_somnolencia.csv") # <-- Reemplaza con el path correcto

# Verificar si hay valores vacíos o nulos

print("Valores nulos por columna:\n", df.isnull().sum())

# Separar variables predictoras y etiqueta

X = df[['EAR', 'MAR', 'Pitch', 'Yaw', 'Roll']]

y = df['Clase']

# División 80/20

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42, stratify=y

)

# Unir X e y antes de guardar

train = pd.concat([X\_train, y\_train], axis=1)

test = pd.concat([X\_test, y\_test], axis=1)

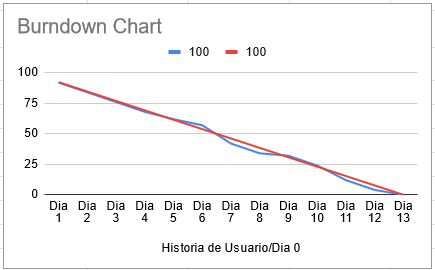
train.to\_csv("train.csv", index=False)

test.to\_csv("test.csv", index=False)

print("Split completado y archivos guardados como train.csv y test.csv")

**5. Burndown Chart**

| Backlog ID | Historia de Usuario | Tiempo Estimado | 08/05 | 09/05 | 10/05 | 11/05 | 12/05 | 13/05 | 14/05 | 15/  05 | 16/  05 | 17/  05 | 18/  05 | 19/  05 | 20/  05 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Dia1** | **Dia2** | **Dia3** | **Dia4** | **Dia5** | **Dia6** | **Dia 7** | **Dia 8** | **Dia 9** | **Dia 10** | **Dia 11** | **Dia 12** | **Dia 13** |
| HU-0.1 | Recolección y Etiquetado del dataset | 8 | 8 | 8 | 4 |  |  |  |  |  |  |  |  |  |  |
| HU-0.2 | Preprocesamiento de datos |  |  |  | 4 | 6 | 5 | **15** |  |  |  |  |  |  |  |
| HU-0.3 | Entrenamiento del Modelo |  |  |  |  |  |  |  |  | 8 | 2 | 8 | **12** | 8 | 4 |
| Tiempo de trabajo | | 100 | 92 | 84 | 76 | 68 | 62 | 57 | 42 | 34 | 32 | 24 | 12 | 4 | 0 |



**6. Análisis del Sprint**

* Se cumplió con todas las tareas estimadas.
* El progreso fue constante y sin bloqueos mayores.
* Se logró un buen ritmo de trabajo.
* Algunas actividades tomaron menos tiempo de lo estimado por lo que se adelantaron 2 actividades del Sprint 0.2.

**7. Conclusiones**

El Sprint 0.1 concluyó exitosamente con todas las historias de usuario. Se hizo una búsqueda dataset para el preprocesamiento de datos, luego se hizo una selección y balanceo de frames por clase, se aumentaron dichos datos, se hizo una selección de algoritmos para luego entrenarlos y validar los resultados para finalmente hacer una selección del algoritmo final.

El equipo demostró capacidad de entrega puntual y buena gestión de riesgos menores. Las métricas indican que la carga fue bien distribuida y los objetivos del sprint se cumplieron completamente.

También se pudo ver que la actividad “estandarización y normalización de datos” no era necesaria, como recomendación, es mejor hacerlos después de la elección del algoritmos para ver si este lo requiere o no.