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:

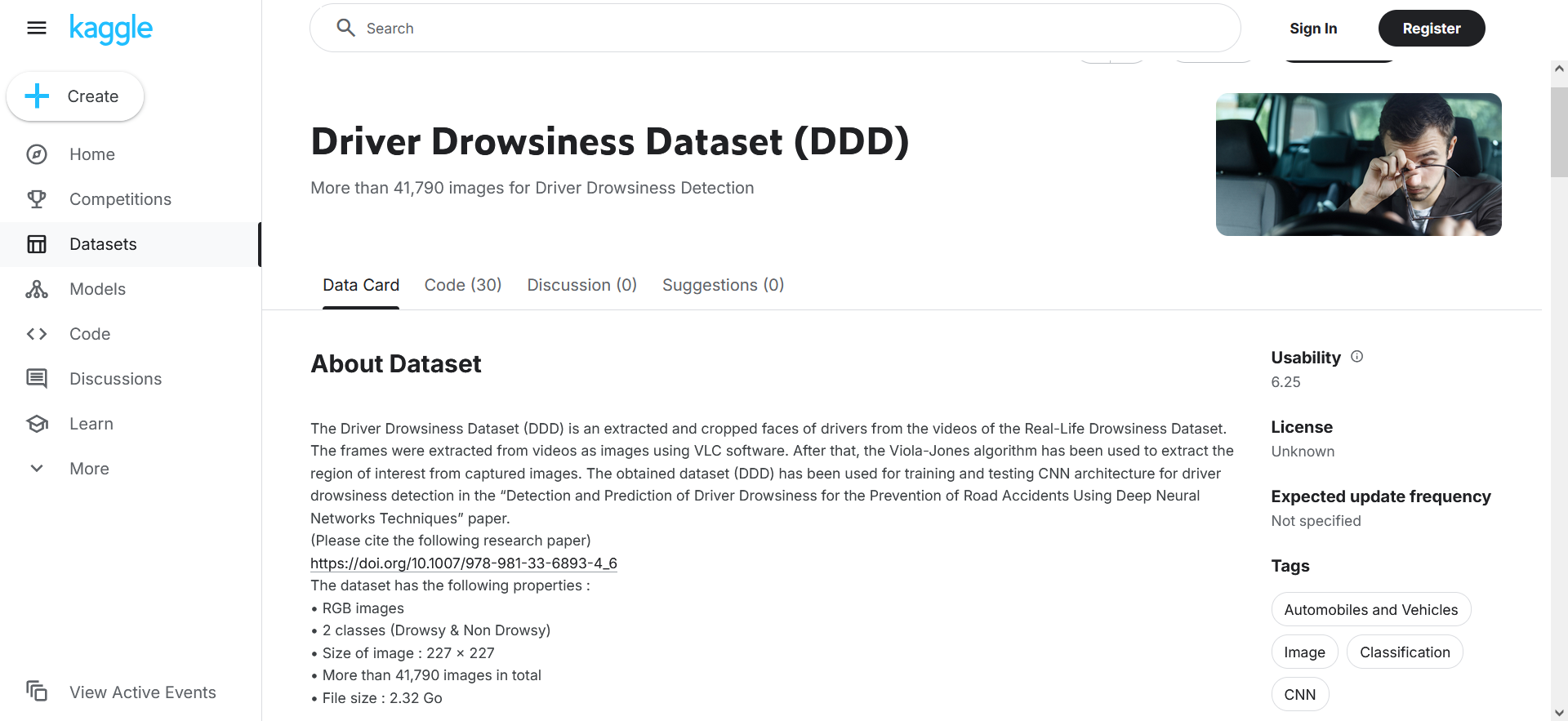
* Usar detección facial en el modelo.
* Monitorear el cierre ocular.

# 3. Historias de Usuario y Tareas

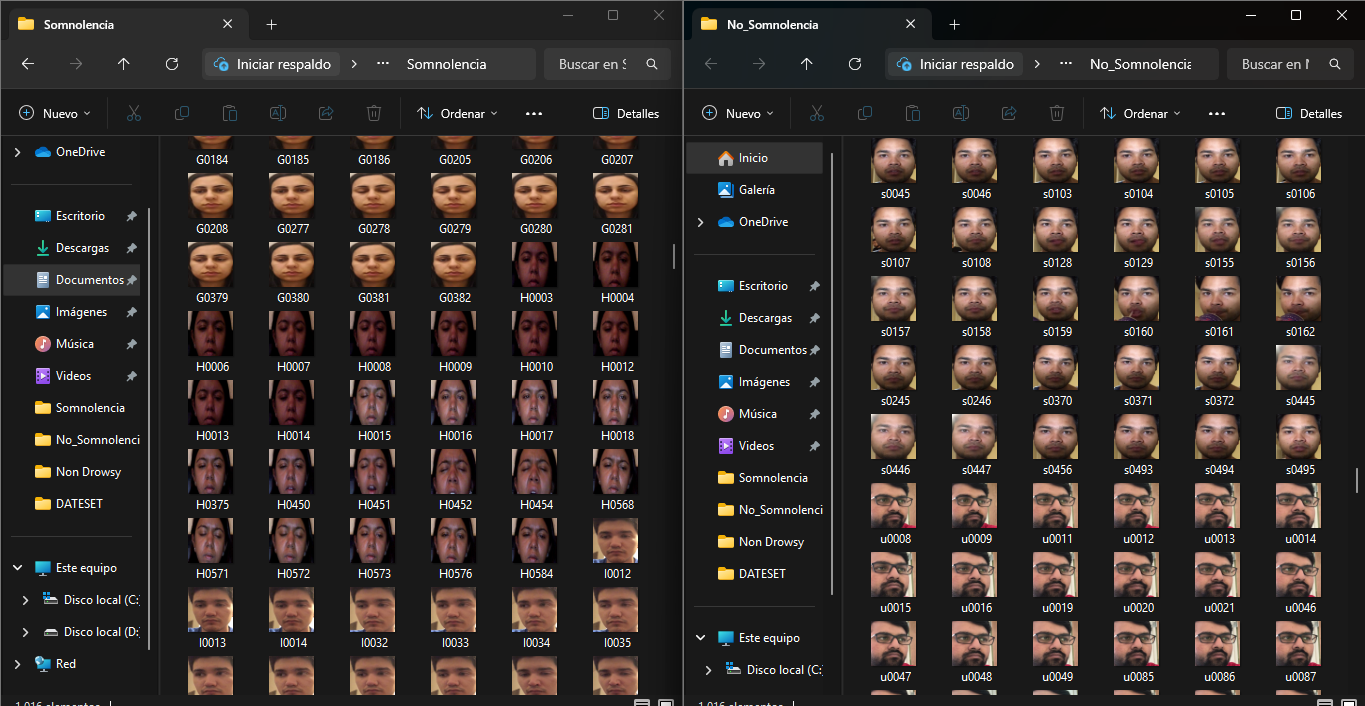
| Backlog ID | Historia de Usuario | Tiempo Estimado | Fecha de Inicio | Fecha de Finalización |
| --- | --- | --- | --- | --- |
| HU-0.1 | Recolección y Etiquetado del dataset | 9 | 08/05/2025 | 10/05/2025 |
| HU-0.2 | Preprocesamiento de datos | 6 | 11/05/2025 | 14/05/2025 |
| HU-0.3 | Entrenamiento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

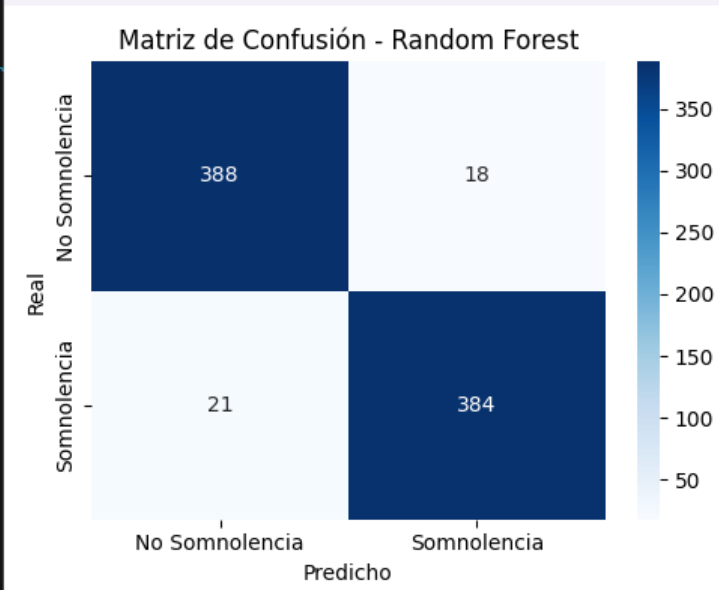
* **Actividad 0.3.1 Seleccionar algoritmo y definir features:** Se definieron los features en base a las métricas recolectadas en los frames del dataset y la selección del algoritmo se hizo con ayuda del análisis de un cuadro comparativo.
  + Features: EAR (Eye aspect ratio), MAR (Mouth aspect ratio), 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 .

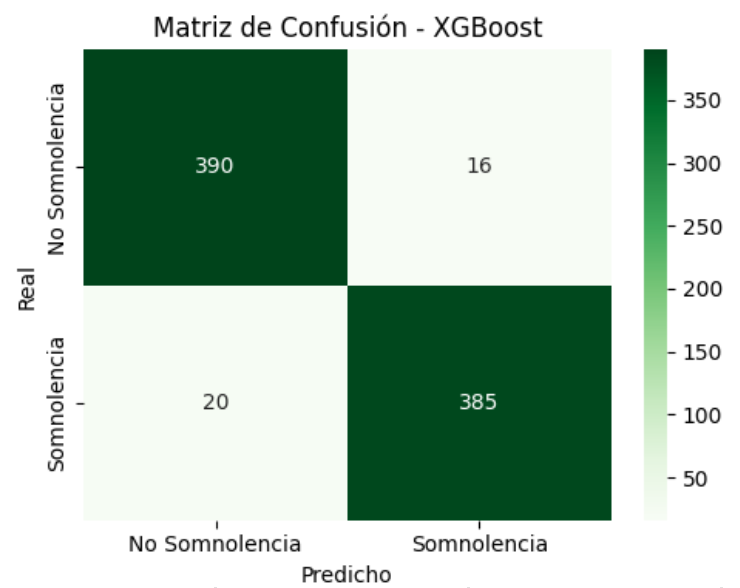
| **RF - Entrenamiento con validación cruzada** | | | |
| --- | --- | --- | --- |
| **Métrica** | **No Somnolencia** | **Somnolencia** | **Accuracy** |
| Precision | 0.93 | 0.95 | 0.94 |
| Recall | 0.95 | 0.93 |  |
| F1-Score | 0.94 | 0.94 |  |
| Support | 1620 | 1621 | 3241 |

| **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 |



En función al resultado de las métricas de rendimiento se puede ver que ambos algoritmos tienen un buen desempeño. Como se puede ver en las matrices de confusión, XGBoost presenta menos falsos positivos y falsos negativos que RF, asimismo, los tests finales, con datos nuevos para el algoritmo después de haber sido entrenado, muestra resultados similares para Recall con 96% y Precision con 95%, el algoritmo XGBoost demostró un F-1 score de 96%, 1% mayor al de RF. Por estos motivos, se escogió el algoritmo XGBoost como algoritmo final para implementación.

## 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

# --------------------- FUNCIONES DE CÁLCULO ---------------------

def euclidean(p1, p2):

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

def compute\_ear(landmarks):

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

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

return ear

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

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

# Puntos de referencia (índices de MediaPipe Face Mesh)

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

# Coordenadas 3D de los landmarks (x, y, z) escalados por el tamaño de la imagen

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")

# Modelo 3D estimado (en mm, basado en proporciones humanas)

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)) # Asumimos 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)

# Comprobación de singularidad

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

# --------------------- CONFIGURACIÓN DE MEDIAPIPE ---------------------

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

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

# --------------------- PROCESAR FRAMES ---------------------

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

# --------------------- RUTAS ---------------------

ruta\_somnolencia = r"C:\Users\USER\Documents\DATESET\DS\_filtrada\Somnolencia"

ruta\_no\_somnolencia = r"C:\Users\USER\Documents\DATESET\DS\_filtrada\No\_Somnolencia"

ruta\_salida = r"C:\Users\USER\Documents\DATESET\metricas\_somnolencia.csv"

# --------------------- EJECUCIÓN ---------------------

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/escalado\_caracteristicas.py Aumentación de datos

import cv2

import os

import numpy as np

import random

# Ajuste de brillo y contraste

def ajustar\_brillo\_contraste(img, brillo=0, contraste=20):

return cv2.convertScaleAbs(img, alpha=1 + contraste / 100.0, beta=brillo)

# Rotación de imagen

def rotar\_imagen(img, angulo):

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

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

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

return cv2.warpAffine(img, M, (w, h))

# Aumentar con aleatoriedad controlada (solo una transformación por imagen)

def aplicar\_augmentacion\_aleatoria(img):

opcion = random.choice(['brillo\_up', 'brillo\_down', 'rotacion'])

if opcion == 'brillo\_up':

return ajustar\_brillo\_contraste(img, brillo=30)

elif opcion == 'brillo\_down':

return ajustar\_brillo\_contraste(img, brillo=-30)

elif opcion == 'rotacion':

angulo = random.choice([-10, 10])

return rotar\_imagen(img, angulo)

return img # fallback

# Procesar una clase

def procesar\_clase(clase\_nombre, input\_base, output\_base):

input\_dir = os.path.join(input\_base, clase\_nombre)

output\_dir = os.path.join(output\_base, clase\_nombre)

os.makedirs(output\_dir, exist\_ok=True)

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

if filename.endswith((".png", ".jpg")):

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

img = cv2.imread(path)

# Aplicar una transformación aleatoria

img\_aug = aplicar\_augmentacion\_aleatoria(img)

# Guardar imagen aumentada

nombre\_salida = f"{os.path.splitext(filename)[0]}\_aug.png"

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

# ---------- RUTAS ABSOLUTAS ----------

input\_base = "/ruta/completa/a/frames\_originales" # contiene 'somnolencia/' y 'no\_somnolencia/'

output\_base = "/ruta/completa/a/frames\_aumentados" # se crearán subcarpetas automáticamente

# ---------- EJECUCIÓN POR CLASE ----------

procesar\_clase("somnolencia", input\_base, output\_base)

procesar\_clase("no\_somnolencia", input\_base, output\_base)

## 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)

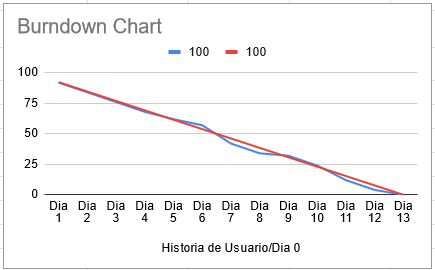
train.to\_csv("train.csv", index=False)

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.