



Facultad de Ingeniería

Diploma en IA - Master en IA

OBLIGATORIO

Clasificación y Detección de Objetos en Imágenes

MATERIA

Machine Learning para Inteligencia Artificial

DOCENTE

Matías Carrasco Piaggio

FECHA

10/07/2025

INTEGRANTES

Reichel Larez - 344474

Natalia Campiglia - 349251

Octavio Revetria - 232745

ÍNDICE

1. INTRODUCCIÓN	3
2. PRE-PROCESAMIENTO	3
3. FEATURE ENGINEERING	4
3.1. PCA (Análisis de Componentes Principales)	4
3.2. HOG (Histograma de Gradientes Orientados)	6
4. CLASIFICADORES	7
4.1. Random Forest	7
4.2. Bagging	10
4.3. Adaptive Boosting	13
4.4. Red Neuronal	15
5. Evaluación y selección de modelos	18
6. Conclusión	23
Anexo	24

1. INTRODUCCIÓN

El objetivo de este trabajo es aplicar técnicas de aprendizaje automático para la clasificación y detección de rostros en imágenes en escala de grises. Se exploran métodos de extracción de características como PCA y HOG, y se entrenan diversos modelos de clasificación, evaluando su desempeño tanto en validación interna como en una competencia de Kaggle.

2. PRE-PROCESAMIENTO

Como parte del preprocesamiento, se construyeron dos conjuntos de imágenes: rostros (“Faces”) y fondos (“Backgrounds”).

Las imágenes de rostros fueron cargadas desde la carpeta Faces utilizando la función `create_images_list`, que recorre todos los archivos `.pgm` y los convierte en un array NumPy. Esto permite manipular las imágenes de manera eficiente para la extracción de características y el entrenamiento de modelos.

```
facesPath = 'Faces'
faces = create_images_list([facesPath])
faces.shape
```

Se cargaron un total de 12.833 imágenes de rostros, cada una de tamaño 64x64 píxeles, lo que da como resultado un array de dimensiones (12833, 64, 64).



De forma similar, las imágenes de fondo (sin rostros) se cargaron utilizando la función **create_images_list**, obteniendo un total de 12.800 imágenes en escala de grises, cada una con una resolución de 64×64 píxeles.



3. FEATURE ENGINEERING

Una vez contruidos los conjuntos de imágenes, se exploraron dos técnicas principales de extracción de características:

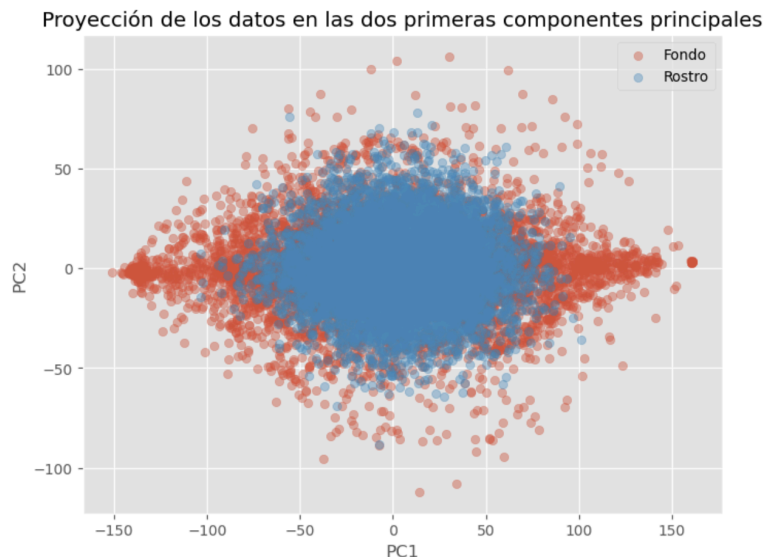
3.1. PCA (Análisis de Componentes Principales)

Para la extracción de características, se utilizó el flattening de las imágenes (conversión de cada imagen 2D en un vector 1D). Posteriormente, los datos fueron normalizados utilizando `StandardScaler` y se aplicó Análisis de Componentes Principales (PCA) para reducir la dimensionalidad. El número de componentes de PCA se seleccionó para explicar al menos el 95% de la varianza. Finalmente, los datos se dividieron en conjuntos de entrenamiento (70%) y test (30%) de manera estratificada y reproducible.

```
X_train_pca, X_test, y_train, y_test, scaler, pca = prepare_data(faces,
backgrounds, test_size=0.3, random_state=42, use_hog=False)
```

El conjunto de entrenamiento resultante contiene X muestras y el de test Y muestras, cada una representada por N componentes principales. Por ejemplo, si se usan 500 componentes y se tiene 17.943 imágenes para entrenamiento y 7.690 para test, los shapes serían:

- `X_train_pca.shape = (17943, 500)`
- `X_test.shape = (7690, 4096)` (antes de aplicar PCA)
- `y_train.shape = (17943,)`
- `y_test.shape = (7690,)`



En el gráfico anterior se muestra la distribución del conjunto de entrenamiento proyectado sobre los dos primeros componentes principales (PC1 y PC2). Los puntos azules corresponden a imágenes de rostros, mientras que los puntos rojos representan fondos. Esta visualización permite observar cómo se agrupan las clases y evaluar su grado de separabilidad en el espacio reducido generado por PCA.

Se puede concluir, entonces, que la mayoría de los datos se agrupan formando una nube alargada a lo largo del eje del primer componente principal (PC1), lo que indica que este concentra gran parte de la información relevante para diferenciar entre rostros y fondos. El segundo componente (PC2), aunque con menor peso, aporta información útil para capturar variaciones internas dentro de cada clase.

Se observa un solapamiento significativo entre ambas clases, especialmente en la zona central, lo que sugiere que PCA, por sí solo, no logra una separación completa. No obstante, la región correspondiente a fondos presenta una mayor extensión, lo que indica que existen áreas del espacio transformado con potencial discriminativo.

En resumen, PCA resulta útil como etapa de reducción de dimensionalidad y detección de patrones globales, pero para mejorar la capacidad de clasificación es necesario combinarlo con técnicas que capturen detalles locales o relaciones no lineales.

3.2. HOG (Histograma de Gradientes Orientados)

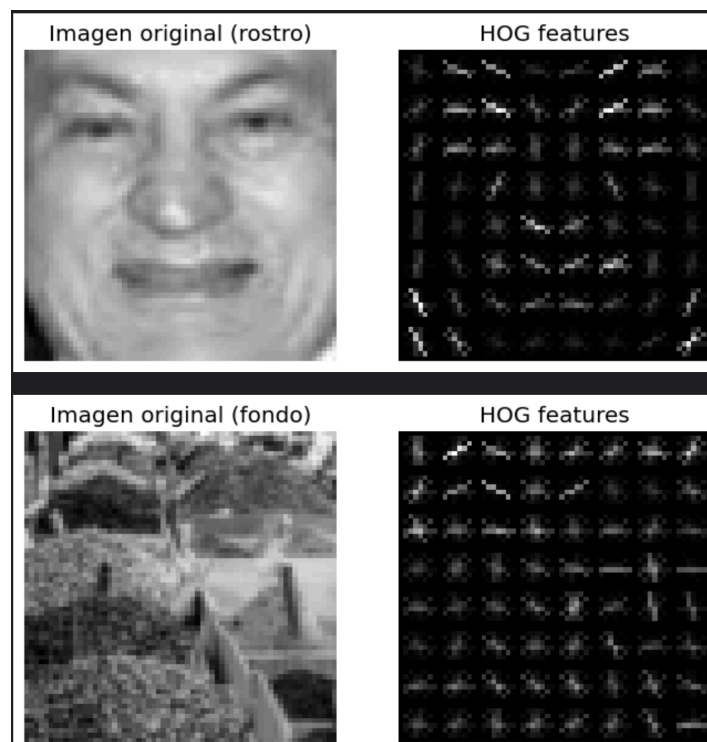
Además de la reducción de dimensionalidad mediante PCA, se exploró la extracción de características HOG (Histogram of Oriented Gradients). HOG permite describir la estructura local de una imagen a partir de los gradientes de intensidad y sus orientaciones, capturando información relevante sobre bordes y texturas.

Para cada imagen, se extrajeron las features HOG utilizando la función **prepare_data** con el parámetro **use_hog=True**. Posteriormente, estas features fueron normalizadas y se aplicó PCA para reducir la dimensionalidad, siguiendo el mismo pipeline que en la técnica anterior.

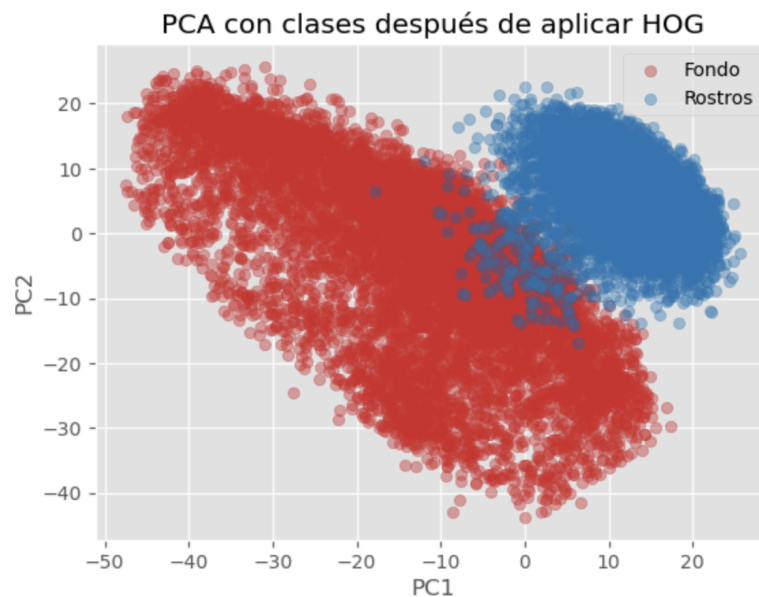
Esta combinación de HOG y PCA permite comparar el desempeño de tres enfoques: únicamente PCA, únicamente HOG y la combinación HOG + PCA.

```
X_train_pca, X_test, y_train, y_test, scaler, pca = prepare_data(  
    faces, backgrounds, test_size=0.3, random_state=42, use_hog=True  
)
```

El uso de HOG permitió obtener un vector de características más robusto para la tarea de clasificación de rostros.



Ahora, al proyectar nuevamente las distribuciones en el espacio de PCA, esta vez utilizando los vectores HOG como entrada, se obtiene el siguiente gráfico:



Esta visualización permite observar el grado de separabilidad entre ambas clases en el espacio reducido generado por la combinación de HOG y PCA. A diferencia de la proyección obtenida utilizando únicamente PCA sobre los píxeles originales, aquí se aprecia una **separación mucho más clara** entre las clases: los rostros (azul) y los fondos (rojo) forman dos grupos bien diferenciados, con un solapamiento mínimo.

En base a los resultados obtenidos, podemos afirmar que la combinación de HOG y PCA no solo reduce la dimensionalidad, sino que también mejora notablemente la separabilidad entre rostros y fondos en el espacio de características. Esto sugiere que el uso de HOG como etapa previa a PCA es altamente beneficioso para la clasificación, ya que permite capturar patrones locales distintivos que no son detectados por PCA aplicado directamente sobre los píxeles.

4. CLASIFICADORES

4.1. Random Forest

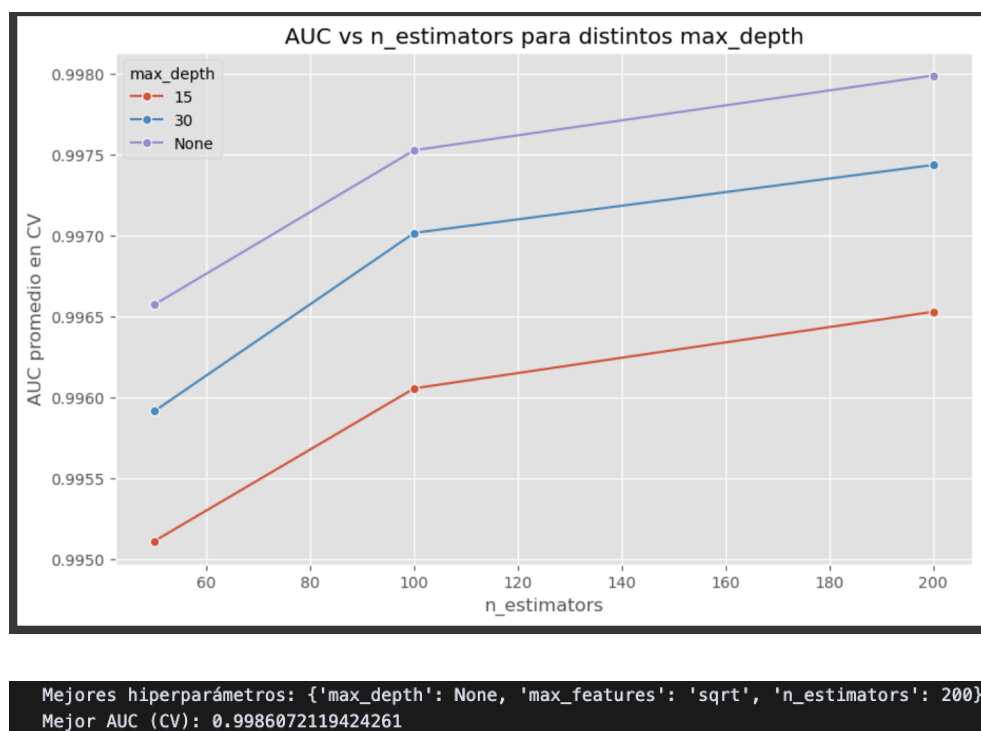
Se exploró el uso de Random Forest como clasificador de referencia por su capacidad para manejar datos de alta dimensionalidad y capturar relaciones no lineales. Random Forest combina múltiples árboles de decisión entrenados sobre

diferentes subconjuntos de datos y características, lo que permite reducir la varianza y mejorar la generalización del modelo.

Para este trabajo, se implementó un proceso de búsqueda de hiperparámetros mediante validación cruzada (GridSearchCV), evaluando combinaciones de parámetros clave como:

- **n_estimators:** número de árboles en el bosque (valores explorados: 50, 100 y 200).
- **max_depth:** profundidad máxima de cada árbol (None, 15 y 30), para controlar el sobreajuste.
- **max_features:** número máximo de características consideradas para dividir en cada nodo (sqrt y log2).

La selección de estos hiperparámetros se realizó utilizando validación cruzada estratificada de 5 folds y se evaluó principalmente con la métrica de AUC, asegurando un buen equilibrio entre precisión y recall para ambas clases.



La gráfica **“AUC vs n_estimators para distintos max_depth”** muestra cómo varía el AUC promedio en validación cruzada para diferentes combinaciones de n_estimators (número de árboles) y max_depth (profundidad máxima de cada árbol).

De esta comparación se desprenden varias observaciones clave:

Incrementar `n_estimators` mejora el AUC promedio. Se observa una tendencia clara: a medida que se incrementa el número de árboles (de 50 a 200), el AUC promedio tiende a subir, indicando que más árboles contribuyen a una mayor estabilidad y precisión del modelo.


La profundidad máxima influye significativamente. Entre los tres valores de `max_depth` evaluados (15, 30 y None), la opción None, que permite a cada árbol crecer sin restricción de profundidad, alcanza consistentemente los valores más altos de AUC. Esto sugiere que en este problema, los árboles más profundos capturan mejor la complejidad de los patrones, aunque debe cuidarse el riesgo de sobreajuste.

De la grafica podemos concluir entonces, que el mejor desempeño se logra con **`n_estimators = 200`**, **`max_depth = None`** y **`max_features='sqrt'`**, donde se alcanza un AUC promedio cercano a 0.998, validando la elección de esta configuración como la mejor para el conjunto de datos.

Una vez determinados los **mejores hiperparámetros**, se entrenó el modelo **Random Forest** con dichos valores, obteniendo la siguiente tabla que muestra los principales **indicadores de desempeño**, incluyendo **precisión**, **recall** y **F1-score** para cada clase (rostros y fondos), tanto en entrenamiento como en prueba. Estos valores permiten evaluar el equilibrio entre aciertos y errores del modelo, su capacidad para distinguir correctamente cada clase y su nivel de generalización sobre datos no vistos.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	8990
1.0	1.00	1.00	1.00	8953
accuracy			1.00	17943
macro avg	1.00	1.00	1.00	17943
weighted avg	1.00	1.00	1.00	17943
	precision	recall	f1-score	support
0.0	0.97	1.00	0.98	3810
1.0	1.00	0.97	0.98	3880
accuracy			0.98	7690
macro avg	0.98	0.98	0.98	7690
weighted avg	0.98	0.98	0.98	7690

Posteriormente, se evaluó el modelo con una **submission en Kaggle**, donde se obtuvo un **resultado de 0.987**, corroborando efectivamente el buen comportamiento y la capacidad predictiva del modelo **Random Forest** en un entorno de prueba independiente.


	submission_RandomForestClassifier.csv Complete · Natalia Campiglia · 6d ago	0.98755
---	--	---------

Random Forest Classifier (200 árboles)

Además, se realizó un entrenamiento adicional del modelo **Random Forest** utilizando **n_estimators = 100**, y sorprendentemente el resultado en Kaggle fue aún mejor que con 200 árboles. Esta diferencia resulta interesante, ya que según el análisis de hiperparámetros basado en validación cruzada, el valor óptimo indicado por el AUC promedio era **n_estimators = 200**.

Una posible explicación es que la métrica utilizada en Kaggle para evaluar las submissions sea el **F1-score**, mientras que el proceso de búsqueda de hiperparámetros se realizó optimizando la métrica **AUC**. Si bien AUC y F1-score suelen estar correlacionadas, no siempre conducen exactamente a la misma configuración óptima, especialmente cuando existe cierto grado de desbalance entre clases o se prioriza un equilibrio particular entre precisión y recall.

Esto pone de manifiesto la importancia de **alinear la métrica de optimización interna con la métrica final de evaluación**, especialmente cuando se trabaja con plataformas externas como Kaggle, donde la forma de medir el desempeño puede afectar la comparación de resultados.

	submission_RandomForestClassifier.csv Complete · Natalia Campiglia · 25d ago	0.99166
---	---	---------

Random Forest Classifier (100 árboles)

4.2. Bagging

Como parte de la comparación de modelos, se entrenó un **BaggingClassifier** utilizando un árbol de decisión como estimador base. Bagging combina múltiples árboles entrenados sobre distintas muestras aleatorias del conjunto de datos, lo que ayuda a reducir la varianza y mejorar la estabilidad de la predicción final.

Para ajustar este modelo, se exploraron distintas combinaciones de hiperparámetros definidos en el espacio de búsqueda:

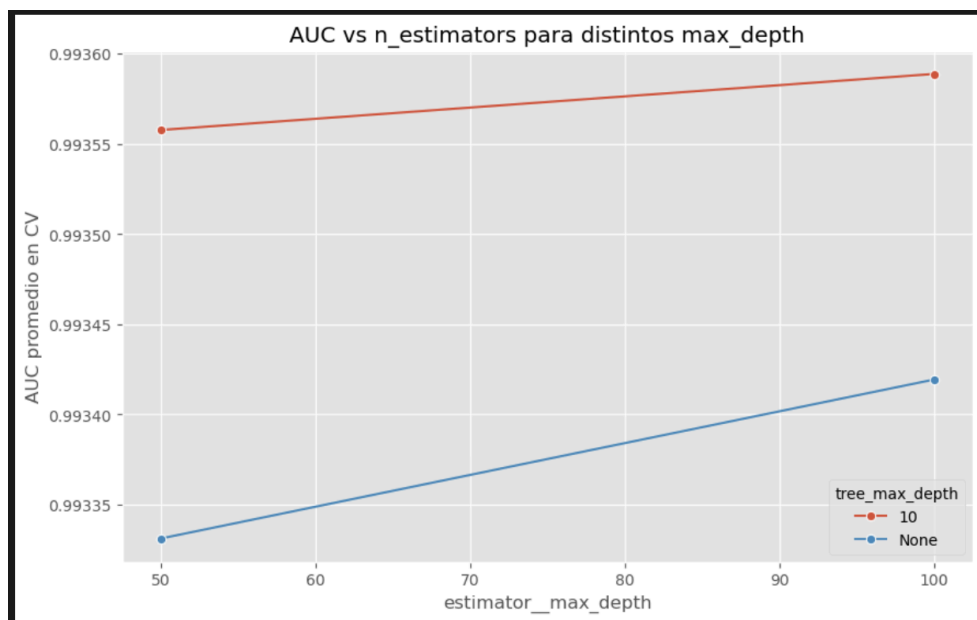
```
param_grid_bagging = {
    'n_estimators': [50, 100],
    'estimator__max_depth': [10, None], # max_depth del árbol base
    'bootstrap': [True, False]
}
```

Donde:

- **n_estimators**: número total de árboles combinados en el Bagging. Más árboles suelen mejorar la estabilidad y reducir la varianza.
- **estimator__max_depth**: controla la profundidad máxima de cada árbol base. Una profundidad limitada ayuda a prevenir el sobreajuste y mejora la generalización.
- **bootstrap**: determina si cada árbol se entrena sobre una muestra aleatoria con reemplazo. Usar bootstrap=True introduce mayor diversidad entre los árboles, lo que suele fortalecer el efecto de bagging.

Mejores hiperparámetros: {'bootstrap': True, 'estimator__max_depth': 10, 'n_estimators': 100}
 Mejor AUC (CV): 0.9977275847083339

	precision	recall	f1-score	support
0.0	0.98	0.99	0.99	3810
1.0	0.99	0.98	0.99	3880
accuracy			0.99	7690
macro avg	0.99	0.99	0.99	7690
weighted avg	0.99	0.99	0.99	7690



La gráfica “AUC vs n_estimators para distintos max_depth” muestra una parte de los resultados obtenidos: revela cómo el AUC promedio en CV tiende a aumentar ligeramente al pasar de 50 a 100 árboles, y cómo limitar la profundidad máxima

de los árboles a 10 genera un rendimiento superior frente a no establecer un límite (None). Esto confirma que controlar la complejidad de cada árbol ayuda a evitar el sobreajuste y mejora la generalización del modelo.

Además, se observa que la opción `bootstrap=True` fue la que condujo a los mejores resultados. Esto es coherente con la lógica de Bagging: usar muestreo con reemplazo introduce mayor diversidad entre los árboles, fortaleciendo la robustez del ensemble y reduciendo la varianza global.

Los mejores hiperparámetros seleccionados fueron: **`bootstrap = True`**, **`estimator__max_depth = 10`** y **`n_estimators = 100`**.

Una vez determinados los **mejores hiperparámetros**, se entrenó el modelo **Bagging** con dichos valores, obteniendo la siguiente tabla que muestra los principales **indicadores de desempeño**, incluyendo **precision**, **recall** y **F1-score** para cada clase (rostros y fondos), tanto en entrenamiento como en prueba. Estos valores permiten evaluar el equilibrio entre aciertos y errores del modelo, su capacidad para distinguir correctamente cada clase y su nivel de generalización sobre datos no vistos.

	precision	recall	f1-score	support
0.0	0.99	1.00	1.00	8990
1.0	1.00	0.99	1.00	8953
accuracy			1.00	17943
macro avg	1.00	1.00	1.00	17943
weighted avg	1.00	1.00	1.00	17943
	precision	recall	f1-score	support
0.0	0.98	0.99	0.99	3810
1.0	0.99	0.98	0.99	3880
accuracy			0.99	7690
macro avg	0.99	0.99	0.99	7690
weighted avg	0.99	0.99	0.99	7690

Podemos ver cómo los resultados tanto en entrenamiento como en testing tienen resultados entre los 0.98 y 0.99.

Posteriormente, se evaluó el modelo con una **submission en Kaggle**, donde se obtuvo un **resultado de 0.937**, un resultado un poco menor a lo esperado, pero

aún así corrobora el buen comportamiento y la capacidad predictiva del modelo **Bagging** en un entorno de prueba independiente.



submission_BaggingClassifier.csv

Complete · Natalia Campiglia · 6h ago

0.93700

4.3. Adaptive Boosting

Siguiendo con la comparación de modelos, se entrenó un AdaBoostClassifier utilizando un árbol de decisión (un árbol de decisión de poca profundidad) como estimador base. AdaBoost es una técnica de boosting que construye un clasificador fuerte combinando secuencialmente múltiples estimadores débiles. Cada estimador se entrena prestando más atención a las instancias que los modelos anteriores clasificaron erróneamente, lo que permite al algoritmo adaptarse al conjunto de datos y mejorar progresivamente la precisión. Este enfoque ayuda a reducir el sesgo y mejora la estabilidad de la predicción final.

Se exploraron distintas combinaciones de hiperparámetros definidos en el espacio de búsqueda:

```
param_grid_ada = {  
    'n_estimators': [50, 100, 200],  
    'learning_rate': [0.5, 1.0, 1.5],  
    'estimator__max_depth': [1, 2, 3]  
}
```

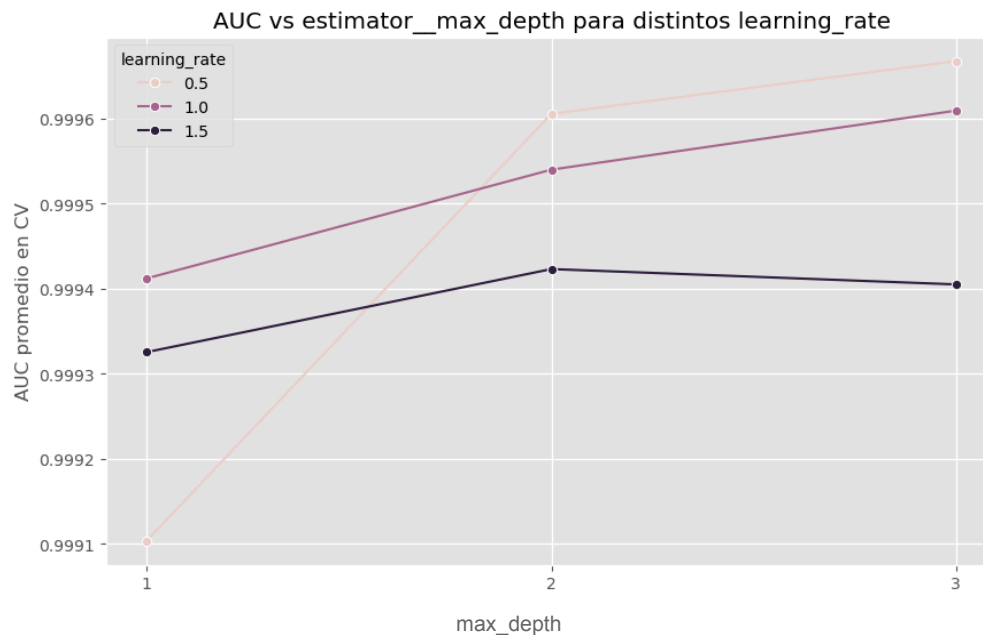
Donde:

- **n_estimators**: número total de árboles entrenados secuencialmente en el modelo. Cuantos estimadores débiles utilizara.
- **learning_rate**: tasa de aprendizaje. Determina el peso de cada árbol en la clasificación final del modelo.
- **estimator__max_depth**: controla la profundidad máxima de cada árbol base. Al tratarse de estimadores débiles tratamos con árboles de baja profundidad.

```
Mejores hiperparámetros: {'estimator__max_depth': 3, 'learning_rate': 1.0, 'n_estimators': 200}
```

```
Mejor AUC (CV): 0.9997226935720382
```

	precision	recall	f1-score	support
0.0	0.99	1.00	1.00	3810
1.0	1.00	0.99	1.00	3880
accuracy			1.00	7690
macro avg	1.00	1.00	1.00	7690
weighted avg	1.00	1.00	1.00	7690



La gráfica muestra una parte de los resultados obtenidos: revela cómo el AUC promedio en CV reacciona a la complejidad de los estimadores base y la tasa de aprendizaje. Se observa que el AUC tiende a mejorar al aumentar la profundidad máxima del estimador base de 1 a 2, indicando que una mayor complejidad de los árboles individuales permite capturar mejor las relaciones en los datos. Sin embargo, al pasar de 2 a 3, el rendimiento se estanca con poca variación, lo que sugiere que limitar la profundidad máxima del estimador base a 2 o 3 genera un rendimiento superior. Esto confirma que controlar la complejidad de cada estimador débil ayuda a evitar el sobreajuste y mejora la generalización del modelo en AdaBoost.

Además, se observa que un learning_rate de 0.5 y 1.0 condujo a los mejores resultados, superando consistentemente a 1.5. Esto es coherente con la lógica de Boosting: un learning rate más pequeño (o moderado) permite que el modelo aprenda de manera más gradual y estable de los errores de los estimadores previos. Esta aproximación más conservadora fortalece la robustez del ensamble y mejora su capacidad de generalización.

Los mejores hiperparámetros seleccionados fueron: **estimator__max_depth = 3**, **n_estimators = 150** y **learning_rate = 1**.

Una vez determinados los **mejores hiperparámetros**, se entrenó el modelo **AdaBoost** con dichos valores, obteniendo la siguiente tabla que muestra los principales **indicadores de desempeño**, incluyendo **precision**, **recall** y **F1-score** para cada clase (rostros y fondos), tanto en entrenamiento como en prueba. Estos valores permiten evaluar el equilibrio entre aciertos y errores del modelo, su capacidad para distinguir correctamente cada clase y su nivel de generalización sobre datos no vistos.

```
Mejores hiperparámetros: {'estimator__max_depth': 3, 'learning_rate': 1.0, 'n_estimators': 150}
Mejor AUC (CV): 0.9996603817491643
```


	precision	recall	f1-score	support
0.0	0.99	1.00	1.00	3810
1.0	1.00	0.99	1.00	3880
accuracy			1.00	7690
macro avg	1.00	1.00	1.00	7690
weighted avg	1.00	1.00	1.00	7690

Podemos ver cómo los resultados tanto en entrenamiento como en testing tienen resultados entre los 0.99 y 1.00.

Posteriormente, se evaluó el modelo con una **submission en Kaggle**, donde se obtuvo un **resultado de 0.975**, un resultado un poco menor a lo visto en testing, pero que valida el buen comportamiento y la capacidad predictiva del modelo **AdaBoost** en un entorno de prueba independiente.

	submission_AdaBoost_Classifier.csv Complete · Natalia Campiglia · 11m ago	0.97540
---	---	----------------

El resultado luego de extender el conjunto de fondos fue:

	submission_AdaBoost_Classifier.csv Complete · Natalia Campiglia · 4h ago	0.99166
---	--	----------------

4.4. Red Neuronal

Como parte del trabajo, se exploraron dos variantes de **redes neuronales** diseñadas y entrenadas con **TensorFlow/Keras**, con el objetivo de comparar su desempeño frente a los modelos de ensemble y evaluar su capacidad de generalización en la tarea de clasificación de rostros y fondos.

Primera variante

La primera arquitectura implementada fue una red neuronal simple y directa, compuesta por dos capas ocultas de 64 y 32 nodos, ambas con función de activación ReLU. Una capa de salida con un nodo y activación sigmoide, adecuada para clasificación binaria. Se utilizó la función de pérdida `binary_crossentropy`, optimizador Adam, y se midió el desempeño con la métrica `accuracy`.

El entrenamiento se realizó durante 50 épocas, con un tamaño de batch de 32.


	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	8990
1.0	1.00	1.00	1.00	8953
accuracy			1.00	17943
macro avg	1.00	1.00	1.00	17943
weighted avg	1.00	1.00	1.00	17943
241/241 ————— 1s 4ms/step				
	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	3810
1.0	1.00	1.00	1.00	3880
accuracy			1.00	7690
macro avg	1.00	1.00	1.00	7690
weighted avg	1.00	1.00	1.00	7690

Los resultados en validación interna mostraron métricas casi perfectas:

Accuracy en test: ~0.9973

Precision, recall y F1-score: ~1.00 en entrenamiento y prueba.

El archivo de submission en Kaggle arrojó un score final de 0.9756, confirmando que, a pesar del rendimiento interno casi perfecto, la generalización final en los datos de evaluación externa muestra una ligera caída.

	submission_NeuralNetwork (1).csv	0.97560
	Complete · Natalia Campiglia · 1d ago	

Segunda Variante

Para reforzar la robustez del modelo, se implementó una segunda versión con una arquitectura más profunda y regularizada, que incluyó:

- Tres capas ocultas de 256, 128 y 64 nodos, todas con activación ReLU.
- Capas Dropout con tasa de 0.3 tras las dos primeras capas ocultas, para reducir el sobreajuste.
- Optimizador Adam con un learning rate reducido (0.0005) para entrenar de forma más estable.
- Función de pérdida `binary_crossentropy`, y métricas de desempeño `accuracy` y `AUC`.
- Implementación de Early Stopping, monitorizando `val_loss` con paciencia de 5 épocas y restaurando los mejores pesos.

Esta configuración se entrenó con un máximo de 100 épocas y `batch_size` de 64.

561/561	1s 2ms/step			
	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	8990
1.0	1.00	1.00	1.00	8953
accuracy			1.00	17943
macro avg	1.00	1.00	1.00	17943
weighted avg	1.00	1.00	1.00	17943
241/241	1s 3ms/step			
	precision	recall	f1-score	support
0.0	0.99	1.00	1.00	3810
1.0	1.00	0.99	1.00	3880
accuracy			1.00	7690
macro avg	1.00	1.00	1.00	7690
weighted avg	1.00	1.00	1.00	7690

Los resultados fueron sobresalientes en validación interna:

Accuracy en test: ~0.9960

AUC en test: ~0.9994

Precisión, recall y F1-score: cercanos o iguales a 1.00 en entrenamiento y prueba.

No obstante, la submission en Kaggle volvió a reflejar un score de 0.9756, muy similar al de la red más simple. Esto sugiere que, si bien una arquitectura más

profunda y regularizada mejora la capacidad de generalización dentro del conjunto de validación, no necesariamente se traduce en una mejora significativa bajo la métrica de evaluación de Kaggle.



submission_NeuralNetwork_earlystopping.csv

Complete · Natalia Campiglia · 16s ago

0.97560

Extender el conjunto de Fondos.

Como prueba adicional, extendemos nuestro conjunto de fondos generando nuevas imágenes sintéticas mediante transformaciones simples (zoom y rotación). Esta estrategia de aumento de datos permitió incrementar la diversidad del set negativo, lo que genera una evaluación más representativa y confiable, reduciendo la varianza de los resultados y permite verificar la verdadera capacidad de generalización del modelo. El aislamiento del conjunto de fondos contribuyó a obtener un score perfecto (1.0) en Kaggle, validando la importancia de un conjunto de entrenamiento balanceado y variado



submission_RedесNeurоnales.csv

Complete · Natalia Campiglia · 16s ago

1.00000

5. Evaluación y selección de modelos

Para la evaluación y selección de modelos se utilizaron múltiples métricas y visualizaciones con el objetivo de comparar objetivamente el desempeño de cada clasificador propuesto (Random Forest, Bagging y Redes Neuronales) y determinar cuál ofrece el mejor balance entre sensibilidad, especificidad y capacidad de generalización.

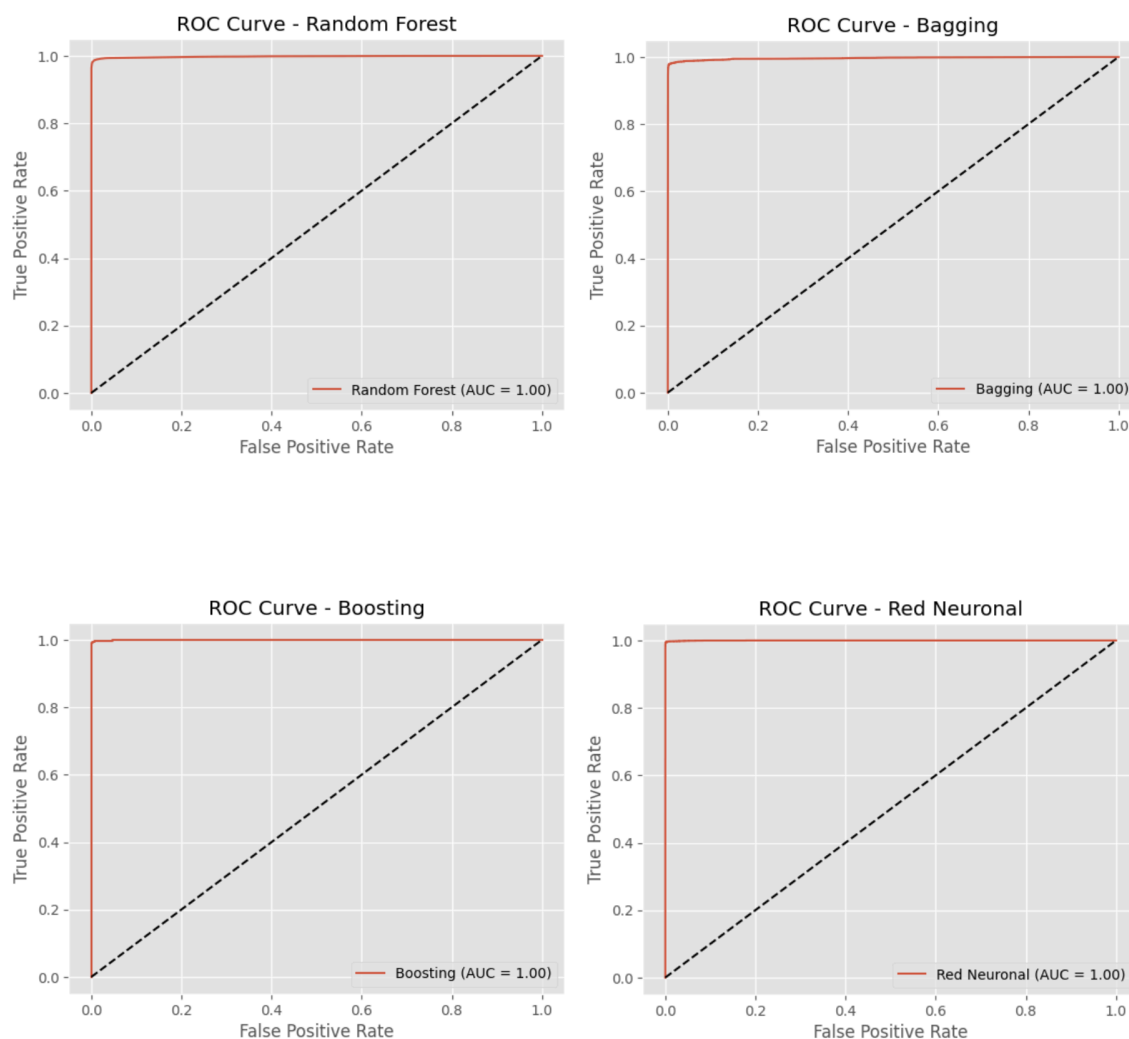
Métricas registradas

Se consideraron las métricas más relevantes para problemas de clasificación binaria, incluyendo:

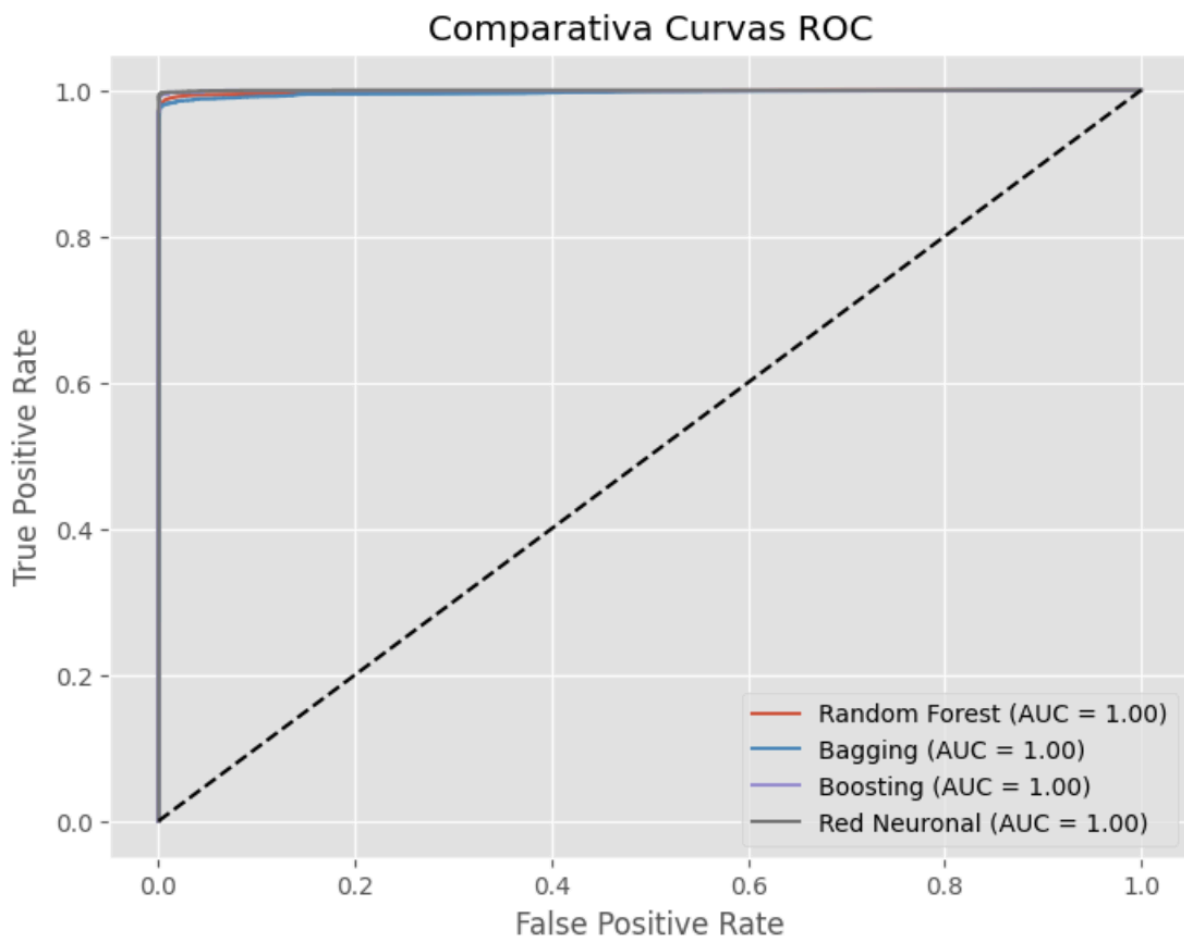
- **TPR (True Positive Rate):** o Recall, mide la proporción de verdaderos positivos correctamente identificados.
- **FPR (False Positive Rate):** proporción de negativos clasificados erróneamente como positivos.
- **Curvas ROC:** se generaron curvas ROC para visualizar la relación TPR-FPR y comparar visualmente la capacidad discriminativa de los modelos.

- **AUC (Área bajo la curva ROC):** resume la capacidad del modelo para distinguir entre clases. Valores cercanos a 1 indican un modelo con excelente discriminación.
- **G-Mean (Geometric Mean):** calcula la media geométrica entre la sensibilidad y especificidad, lo que permite valorar de forma balanceada la precisión para ambas clases, especialmente útil en contextos con clases desbalanceadas.
- **Training Time:** Se calcula el tiempo en segundos de la demora del modelo en hacer el entrenamiento.

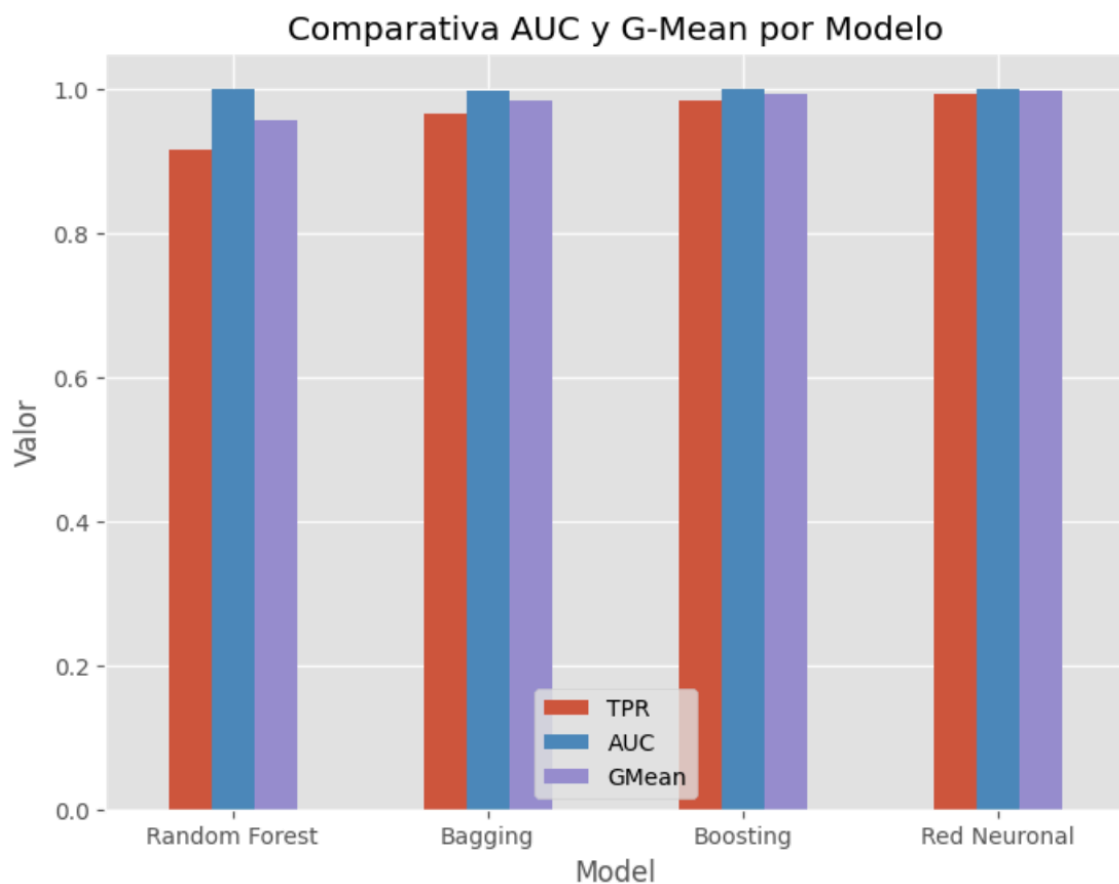
Cada modelo se evaluó en conjunto de prueba con un `classification_report` detallado.



Podemos apreciar cómo todas las curvas ROC se encuentran muy próximas al vértice superior izquierdo del gráfico. Esto implica que los modelos logran una alta TPR con una FPR mínima, lo cual es ideal para una tarea de clasificación binaria.



En esta gráfica observamos las curvas ROC de todos los modelos a la vez, las curvas prácticamente se solapan, aunque podemos ver cómo la curva de la red neuronal es la que tiene un mejor desempeño, mientras que la curva de bagging es la que más se aleja del vértice superior izquierdo. Por otro lado, Random Forest parece estar entre medio de la curva de la Red Neuronal y la de Bagging, y la curva de Boosting se encuentra escondida, probablemente debajo de la curva de la red neuronal.



	Model	TPR	FPR	AUC	GMean	TrainingTime
0	Random Forest	0.9152	0.0000	0.9981	0.9567	504.8419
1	Bagging	0.9659	0.0007	0.9965	0.9825	3358.4592
2	Boosting	0.9836	0.0002	0.9996	0.9917	2796.2570
3	Red Neuronal	0.9925	0.0005	0.9998	0.9960	325.8412

Análisis de las métricas obtenidas:

- **TPR y FPR balanceados**

- La Red Neuronal muestra el TPR más alto (0.9925) y mantiene una FPR extremadamente baja (0.0005), logrando un equilibrio ideal.
- Bagging y Boosting también alcanzan TPR altos (>0.96) con FPR mínimos, lo que refleja buena sensibilidad sin sacrificar especificidad.

- En comparación, el Random Forest tiene el TPR más bajo (0.9152) aunque su FPR es cero absoluto, lo que indica extrema precaución para evitar falsos positivos, pero a costa de perder algunos verdaderos positivos.
- **Precisión discriminativa (AUC)**
 Todos los modelos alcanzan valores de AUC superiores a 0.996, lo que indica una altísima capacidad para distinguir entre clases. Destaca el Boosting con un AUC de 0.9996 y la Red Neuronal con 0.9998, evidenciando que estos enfoques logran capturar muy bien los patrones presentes en los datos.
- **G-Mean como métrica de balance**
 El G-Mean combina sensibilidad y especificidad y muestra un patrón coherente:
 - Boosting (0.9917) y Red Neuronal (0.9960) lideran con valores casi perfectos.
 - Bagging (0.9825) mantiene un balance muy alto.
 - Random Forest destaca por su FPR = 0, pero su menor TPR reduce el G-Mean a 0.9567, situándose como el menos balanceado en términos de detección y error.
- **Eficiencia computacional (Tiempo de entrenamiento)**
 - Random Forest y la Red Neuronal simple fueron los modelos más rápidos de entrenar, requiriendo aproximadamente 505s y 325s respectivamente.
 - Por el contrario, Bagging (~3358 segundos) y Boosting (~2796 segundos) requieren tiempos significativamente mayores, debido al número de estimadores y la complejidad de combinar múltiples árboles base.

Kaggle results

	Random Forest	Bagging	Boosting	Redes Neuronales
Kaggle Score	0.99166	0.93700	0.99166	1.00000

6. Conclusión

En este trabajo se exploraron distintas técnicas de preprocesamiento y extracción de características, como PCA y HOG, combinadas con modelos de clasificación robustos: Random Forest, Bagging, Boosting y Redes Neuronales, evaluando su desempeño de forma integral mediante métricas clave (TPR, FPR, AUC, G-Mean), curvas ROC, tiempos de entrenamiento y resultados finales en Kaggle.

Los resultados evidencian que todos los modelos alcanzaron una capacidad discriminativa sobresaliente ($AUC > 0.996$) y una excelente relación TPR–FPR, demostrando la efectividad del pipeline implementado. Entre ellos, la Red Neuronal, incluso con una arquitectura sencilla, se destacó por lograr un Kaggle Score perfecto (1.0) tras ampliar y diversificar el conjunto de fondos mediante técnicas de aumento de datos, validando la importancia de contar con un conjunto de entrenamiento balanceado y representativo.

En cuanto a la eficiencia computacional, Random Forest y la Red Neuronal resultaron ser las opciones más rápidas de entrenar y, además, obtuvieron resultados sólidos y consistentes en Kaggle. Por el contrario, Bagging demostró ser la opción menos conveniente, ya que combinó un alto costo computacional con los peores resultados en Kaggle, lo que evidencia que, para este problema en particular, su uso no resulta eficiente ni justificado frente a alternativas más robustas y rápidas como Random Forest o Boosting.

Estos hallazgos subrayan la importancia de contar con un conjunto de prueba lo suficientemente amplio para reducir la varianza y asegurar resultados robustos frente a datos no vistos. Los puntajes finales en Kaggle confirmaron la validez de este enfoque, mostrando que los modelos bien balanceados y ajustados logran una mayor capacidad de generalización.

Finalmente, este trabajo demuestra que la combinación de un buen preprocesamiento, técnicas de aumento de datos y una validación rigurosa permite construir clasificadores robustos y confiables para tareas de detección de rostros, sentando bases sólidas para futuras mejoras mediante arquitecturas más profundas y estrategias de regularización avanzadas.

Anexo

Por más información sobre la implementación del proyecto consulta nuestro repositorio de github <https://github.com/nataliacampiglia/ml-ia-obligatorio>