

Universidad ORT Uruguay

Facultad de Ingeniería Ing. Bernard Wand-Polak

Documentación de Obligatorio - Grupo M6A

Machine Learning

Andrés Naistat-Hauser - 273470

Nicolás Cababie - 203656

Santiago Morón – 267553

Link al repositorio: <https://github.com/Santiago-Moron/MachineLearning-OBL1>

Índice

Abstract.....	3
Preprocesamiento:.....	3
Extracción de features:.....	3
Algunas consideraciones:.....	3
Clasificadores:.....	4
Clasificadores utilizados:.....	4
Ada Boost.....	4
Bagging.....	4
Random Forest.....	5
Regresión Logística.....	5
Gradient Boosting.....	5
KNN.....	5
Red neuronal.....	5
Primer modelo:.....	6
Segundo modelo:.....	6
Tercer modelo:.....	6
Cuarto modelo:.....	7
Modelo final:.....	7
Evaluación de modelos:.....	9
Elección de modelo final:.....	12
Métricas finales en datos de test:.....	13
Conclusión:.....	14
Bibliografía:.....	15

Abstract

En este informe podemos encontrar paso a paso cuál fue la evolución del obligatorio, este es un reporte de los resultados con explicaciones y fundamentos de las decisiones tomadas, no se documenta específicamente el código que se desarrolló. Para evaluar el código, lo puede encontrar en el repositorio, siendo el notebook de Python el archivo “ObligatorioML.ipynb”, el resto de archivos fueron útiles para cargar modelos ya entrenados y ahorrar tiempo durante el desarrollo, pero no son relevantes para el producto final.

Preprocesamiento:

El preprocesamiento fue muy importante debido al volumen de datos de entrada que tenemos en este obligatorio al usar el dataset de “lfw_people”, en donde tenemos más de 13.000 imágenes. El primer paso realizado en el obligatorio fue leer la documentación de LFW Dataset, algunos aportes de esta lectura fue que existen valores de cross-validation recomendados, se explica la cantidad de personas que tienen más de una imagen y conocemos la resolución de las imágenes.

Ahora comenzando con nuestro labor, originalmente usamos directamente este dataset para extraer las HOG Features, pero para conocer distintas opciones, terminamos debatiendo cómo podíamos aliviar el trabajo computacional de extraer features sin perder información en exceso.

Para eso, decidimos reajustar el tamaño de las imágenes como se recomendó en clase. Este proceso hace que se pierda información respecto a la imagen original, pero concluimos que los rostros siguen teniendo características suficientemente distinguibles como para poder identificarlos de todas maneras y hace al procesamiento más liviano.

También planteamos como una opción válida de aplicar antialiasing y reescalar la resolución de las imágenes, pero decidimos no hacerlo para no perder aún más información.

Se trató de usar la función ProfileReport para analizar la correlación de las variables, proceso que fue muy extenso debido a la cantidad de datos, pero no arrojó resultados reveladores ya que no tenemos distintas variables a estudiar sino que solo arrays de números que representan cada píxel.

Extracción de features:

La extracción de features la logramos utilizando la extracción de HOG Features vista en clase (a pesar de que se sugiere usar Haar Features, este cambio fue validado por los docentes). En esta función, extraemos parches de la imagen para un tamaño definido por parámetro “size” y luego se extraen las features de cada parche.

Algunas consideraciones:

- Para este trabajo se tomó en cuenta la paralelización en múltiples “jobs” para los modelos que llevaban mucho tiempo en ser entrenados, por la cantidad de estimadores, esto fue utilizado en el Random Forest y también durante algunas de las pruebas con redes neuronales.
- Se guardaron modelos usando las funciones “Dump” y “Load” de la librería “joblib”

- para no tener que re-entrenarlos cada vez que abría el proyecto.
- Se usó un historial con la función “log_dir” durante el desarrollo de la red neuronal, como no fue utilizado en demasía, no se extendió su uso a otros modelos.

Clasificadores:

Para poder realizar la técnica de combinación de clasificadores, conocida como “ensembles”, se decidió priorizar en la construcción de aquellos clasificadores que son buenos en detectar aquellas imágenes que no son rostros. De esta manera, se descartan aquellas imágenes que no nos serán útiles y las demás serán derivadas al siguiente modelo para seguir clasificando. Con esto, se quiere intentar maximizar el TNR (true negative rate) con el fin de poder descartar las imágenes que no son rostros antes de que sean analizadas por clasificadores más complejos.

Clasificadores utilizados:

Ada Boost

Este modelo consiste en construir un clasificador fuerte a partir de clasificadores débiles, ajustando los pesos de las instancias de los datos con el objetivo de darle más importancia a los datos mal clasificados.

Como clasificador débil se eligió utilizar un árbol de decisión de profundidad 1, para optimizar el tiempo de ejecución respecto a clasificadores más complejos. El número de estimadores también afecta el tiempo de ejecución, aunque un mayor número de éstos lleva a un mejor rendimiento.

Bagging

El clasificador de Bagging utiliza un clasificador base con el objetivo de iterar sobre él con distintos set de datos para reducir la varianza y mejorar la generalización del modelo. Bagging utiliza muestreo con reposición para generar diferentes conjuntos de datos para cada iteración del clasificador base.

El clasificador base puede ser cualquier modelo de clasificación o regresión. Es complicado elegir un clasificador base ya que si es demasiado complejo puede llevarnos al sobreajuste. En un principio se utilizó Ada Boost como clasificador base y luego por un momento se terminó eligiendo a un árbol de decisión de profundidad 2.

Esto ocurrió por diversos motivos: los árboles de decisión son más simples y fáciles de interpretar, también ayudan a evitar el sobreajuste y son más eficientes en términos de ejecución. Luego al agregar el clasificador de Random Forest en el ensemble, nos dimos cuenta de que usar Bagging con árboles de decisión como clasificador base no iba a ser muy útil ya que quedaría un comportamiento muy similar al Random Forest.

Finalmente, se eligió usar como clasificador base a la regresión logística. Creemos que es útil ya que se utiliza comúnmente en problemas de clasificación binaria (en nuestro caso, poder decir si una imagen contiene una cara o no).

Random Forest

La idea principal de este clasificador es combinar varios árboles de decisión. La principal diferencia que tiene con Bagging es que se usan únicamente árboles de decisión como clasificador base mientras que en Bagging se puede usar cualquier modelo.

Para cada conjunto de entrenamiento, se entrena un árbol de decisión. Cada árbol se entrena utilizando una muestra aleatoria de las instancias y características disponibles. Esto introduce variabilidad y diversidad en los árboles individuales.

Vemos útil a este clasificador ya que tiene una gran capacidad para manejar grandes conjuntos de datos.

Regresión Logística

Como se mencionó anteriormente, la regresión logística se utiliza con frecuencia en los problemas de clasificación binaria, donde el espacio de etiquetas tiene dos opciones. Es un modelo fácil y rápido de entrenar.

Gradient Boosting

Este clasificador tiene un comportamiento similar a Ada Boost aunque difieren en algunos aspectos. A diferencia de Ada Boost que ajusta los pesos de los datos para mejorar la clasificación, este modelo aprende de los errores de predicción ajustando los residuos en busca de mejorar el ajuste del modelo.

KNN

Busca los "K" puntos de datos más cercanos al nuevo punto y toma la mayoría (en clasificación) o promedio (en regresión) de sus valores para predecir la clase o el valor del nuevo punto. Es como preguntar a tus vecinos más cercanos qué piensan para tomar decisiones.

Red neuronal

Para realizar la red neuronal se tomaron en cuenta muchas variaciones distintas para poder alcanzar un modelo aceptable, el primer paso que se tuvo que definir fue el modelo de capas a utilizar. Comenzamos con 3 capas densas, la primera de 8 neuronas, la siguiente de 5 y la última de 1. La segunda de las pruebas fue que la segunda capa tuviera más neuronas que el resto, para poder extraer nuevas features de las capas anteriores.

También experimentamos con distintas funciones de activación incluso usándolas en conjunto, la versión que más utilizamos fue que las primeras dos capas utilizaran funciones Relu mientras que la última es una función Sigmoidea.

Otro hiper parámetro que varió fue el Batch-size, comenzando con el valor predeterminado de 10, luego siguiendo las sugerencias de los docentes, se evaluó el modelo con 32 batches y finalmente con 128. Otra de las variaciones fue en base al optimizador usado, encontramos que existe 'SGD', 'RMSprop', 'Adagrad', 'Adadelata', 'Adam', 'Adamax', 'Nadam'. Debido al prolongado tiempo que conlleva entrenar y evaluar un nuevo modelo, solo probamos con SGD y Adam.

También se probó con dos epochs distintas, primero con 100 y luego con 50.

Respecto a los regularizadores, originalmente se utilizó “Early-Stopping”, en la versión final se sustituyó por Dropout para “apagar neuronas” y reducir el sobreajuste. También probamos el modelo con “Data Augmentation”, que uno de sus casos de uso típicos es el análisis de imágenes.

Una de las dificultades a la hora de codificar la red neuronal fueron los problemas de compatibilidad entre SKlearn y Tensorflow, ya que no se podían usar directamente las funciones de Cross-Validation de la librería de SKlearn en nuestro modelo. Esto se solucionó tras investigar acerca de los “Wrappers”, usamos la función KerasClassifier para crear un modelo Keras.

Para evaluar el resultado final se usaron dos métricas, la matriz de confusión y la gráfica Validation Error/Train Error.

A continuación presentaremos algunos de los resultados obtenidos en distintas versiones de redes neuronales:

Primer modelo:

12, 8 y 1 neurona por capa, optimizador Adam, Batch Size 10, 150 Epochs. Se usa Early-Stopping.

```
Recall: 0.9923334930522281
True Negative Rate (TNR): 0.9955686853766618
```

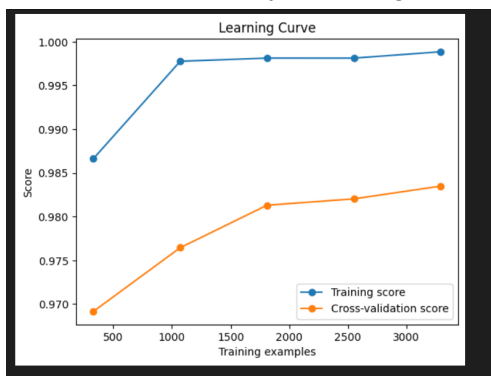
Segundo modelo:

8, 12 y 1 neurona por capa, optimizador SGD, Batch Size 32, 50 Epochs. Se usa Early-Stopping.

```
Recall: 0.9904168663152851
True Negative Rate (TNR): 0.9847365829640571
```

Tercer modelo:

6, 8 y 1 neurona por capa, optimizador SGD, learning rate 0,01, Batch Size 128, 300 Epochs. Se usa Early-Stopping.

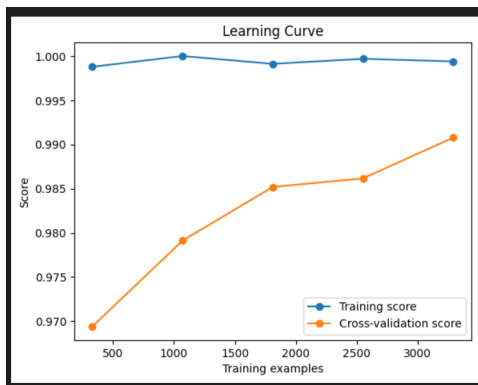


Recall: 0.9870627695256349
True Negative Rate (TNR): 0.9891678975873953

En este punto notamos que los resultados son mejores con un Epoch más alto. Se seguirá incrementando el Epoch, además de que se intentará agregar más capas a la red. También se incrementará el Learning Rate de 0,01 a 0,05.

Cuarto modelo:

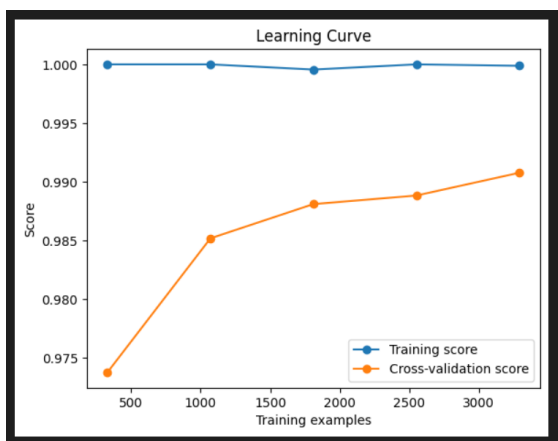
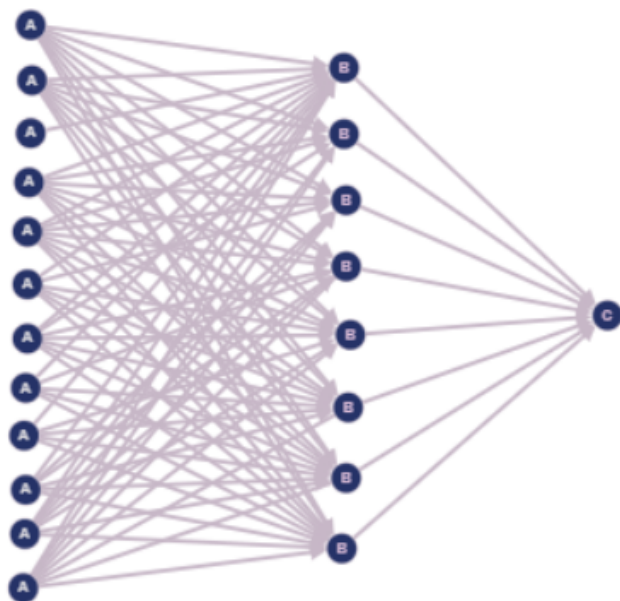
6, 8, 20, 8 y 1 neurona por capa, aplicando Dropout, optimizador SGD, learning rate 0,01, Batch Size 128, 300 Epochs. Se usa Early-Stopping.



Recall: 0.9856252994729277
True Negative Rate (TNR): 0.9950763170851797

Modelo final:

Por Navaja de Occam, decidimos usar un modelo menos complejo ya que todos arrojaron resultados aceptables, esta última versión solo tiene tres capas, con 12, 8 y 1 neurona respectivamente (igual que en el primer modelo). Se ajustaron los hiper parámetros como la tasa de aprendizaje, se agregó Dropout, en dos de las capas, se utiliza activación Relu en las primeras dos y Sigmoides en la última capa, esto último sustentado en que estamos frente a un problema de clasificación binaria, por lo que la última función de activación debe ser Sigmoides y con Loss BCE.

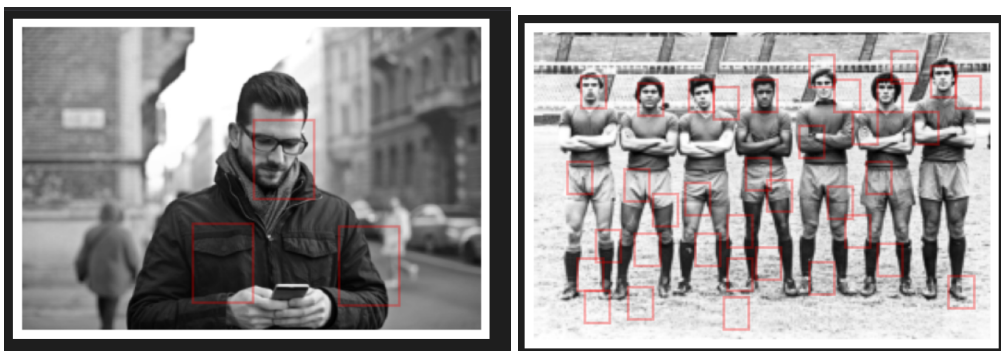


```
[[[2017, 14],
 [ 26, 2061]],
```

Recall: 0.9875419262098706

True Negative Rate (TNR): 0.9931068439192516

Con este modelo final, podemos ver los resultados en dos imágenes de ejemplo, el código para realizar esto fue el mismo que el visto en clase:



Como podemos ver este modelo detecta bien los rostros pero también tiene alucinaciones, detectando rostros donde no los hay. Esto no es una gran preocupación en este punto del obligatorio ya que nos centramos en que tenga un TNR alto, es decir, que cuando descarte

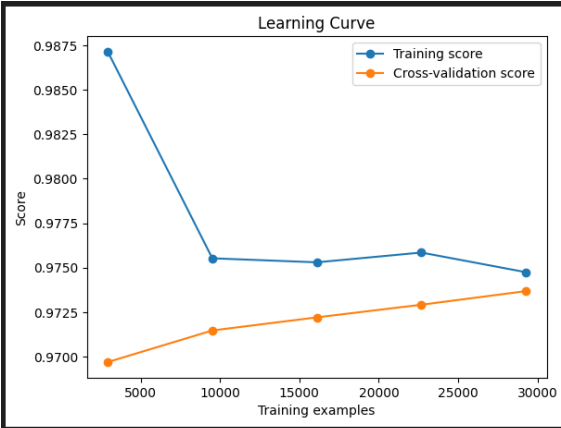
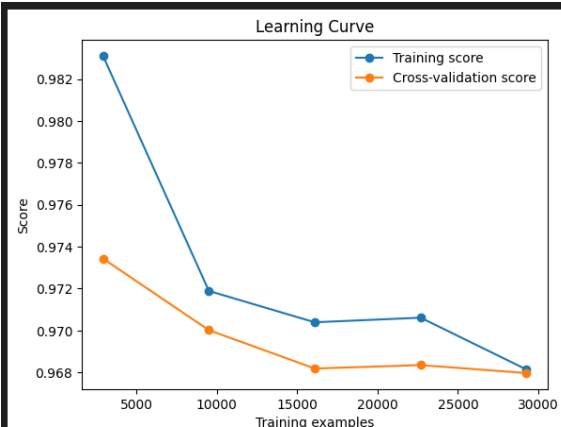
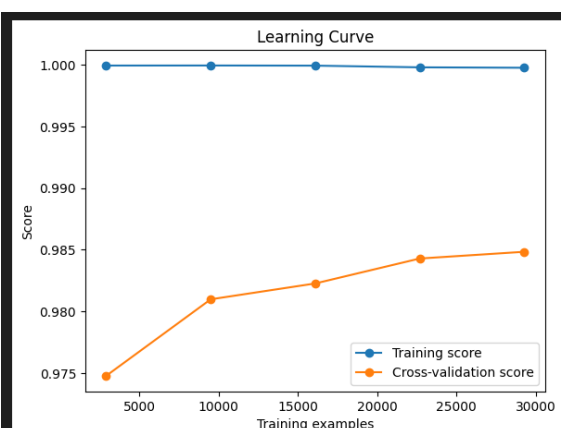
un parche, que de verdad no haya ningún rostro en ese parche. Aquí podemos ver que logra hacer eso a la perfección, no descartó ninguno de los rostros.

Para probar su funcionamiento, además de las métricas anteriores, usamos una cross-validation con 10 folds, como lo sugiere la documentación de `lfw_people()`.

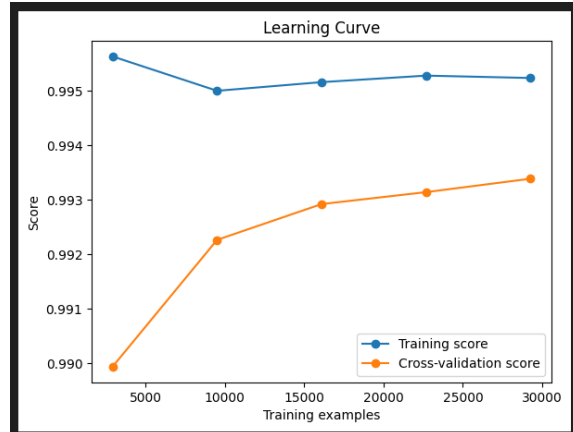
Un último comentario en esta sección de red neuronal puede ser sobre los tipos de regularización que no utilizamos, pero merecen una mención ya que los tomamos en cuenta a la hora de hacer pruebas para encontrar el mejor modelo. Consideramos usar una regularización de “Data-Augmentation” ya que es una de las recomendadas para análisis de imágenes, además de “Batch-Normalization” que puede ser útil para hacer más rápido el entrenamiento. También se contempló usar Warm-Start pero se decidió no complejizar demasiado el modelo final.

Evaluación de modelos:

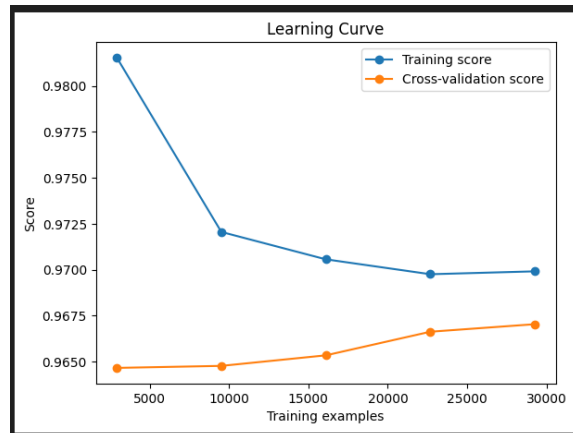
	Parámetros	Accuracy	Precisión	Recall	TNR	TPR	FPR
Ada Boost	estimators = 20	0.970	0.953	0.945	0.981	0.945	0.019
Bagging	estimators = 10	0.970	0.963	0.927	0.985	0.927	0.014
Random Forest	estimators = 500	0.969	0.963	0.927	0.985	0.927	0.014
Regresión Logística	iter = 500	0.992	0.992	0.982	0.997	0.982	0.003
Gradient Boosting	estimators = 20	0.964	0.990	0.882	0.996	0.882	0.003
KNN	neighbors = 15	0.995	0.998	0.983	0.999	0.983	0.000
Red neuronal	sigmoid, relu, batch = 128 epochs = 150	0,990	0,986	0,98	0,993	0,993	0,007

Clasificador	Curva																		
Ada Boost	<div><p>Learning Curve</p><table><thead><tr><th>Training examples</th><th>Training score</th><th>Cross-validation score</th></tr></thead><tbody><tr><td>3000</td><td>0.987</td><td>0.970</td></tr><tr><td>10000</td><td>0.976</td><td>0.971</td></tr><tr><td>16000</td><td>0.976</td><td>0.972</td></tr><tr><td>23000</td><td>0.976</td><td>0.973</td></tr><tr><td>30000</td><td>0.975</td><td>0.974</td></tr></tbody></table></div>	Training examples	Training score	Cross-validation score	3000	0.987	0.970	10000	0.976	0.971	16000	0.976	0.972	23000	0.976	0.973	30000	0.975	0.974
Training examples	Training score	Cross-validation score																	
3000	0.987	0.970																	
10000	0.976	0.971																	
16000	0.976	0.972																	
23000	0.976	0.973																	
30000	0.975	0.974																	
Bagging	<div><p>Learning Curve</p><table><thead><tr><th>Training examples</th><th>Training score</th><th>Cross-validation score</th></tr></thead><tbody><tr><td>3000</td><td>0.983</td><td>0.973</td></tr><tr><td>10000</td><td>0.972</td><td>0.970</td></tr><tr><td>16000</td><td>0.970</td><td>0.968</td></tr><tr><td>23000</td><td>0.970</td><td>0.968</td></tr><tr><td>30000</td><td>0.968</td><td>0.968</td></tr></tbody></table></div>	Training examples	Training score	Cross-validation score	3000	0.983	0.973	10000	0.972	0.970	16000	0.970	0.968	23000	0.970	0.968	30000	0.968	0.968
Training examples	Training score	Cross-validation score																	
3000	0.983	0.973																	
10000	0.972	0.970																	
16000	0.970	0.968																	
23000	0.970	0.968																	
30000	0.968	0.968																	
Random Forest	<div><p>Learning Curve</p><table><thead><tr><th>Training examples</th><th>Training score</th><th>Cross-validation score</th></tr></thead><tbody><tr><td>3000</td><td>1.000</td><td>0.975</td></tr><tr><td>10000</td><td>1.000</td><td>0.981</td></tr><tr><td>16000</td><td>1.000</td><td>0.982</td></tr><tr><td>23000</td><td>1.000</td><td>0.984</td></tr><tr><td>30000</td><td>1.000</td><td>0.985</td></tr></tbody></table></div>	Training examples	Training score	Cross-validation score	3000	1.000	0.975	10000	1.000	0.981	16000	1.000	0.982	23000	1.000	0.984	30000	1.000	0.985
Training examples	Training score	Cross-validation score																	
3000	1.000	0.975																	
10000	1.000	0.981																	
16000	1.000	0.982																	
23000	1.000	0.984																	
30000	1.000	0.985																	

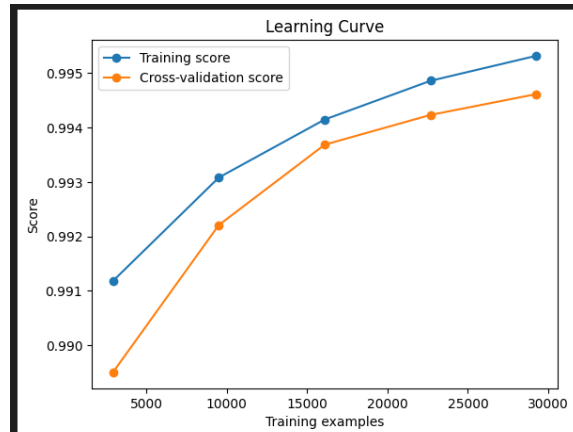
Regresión Logística

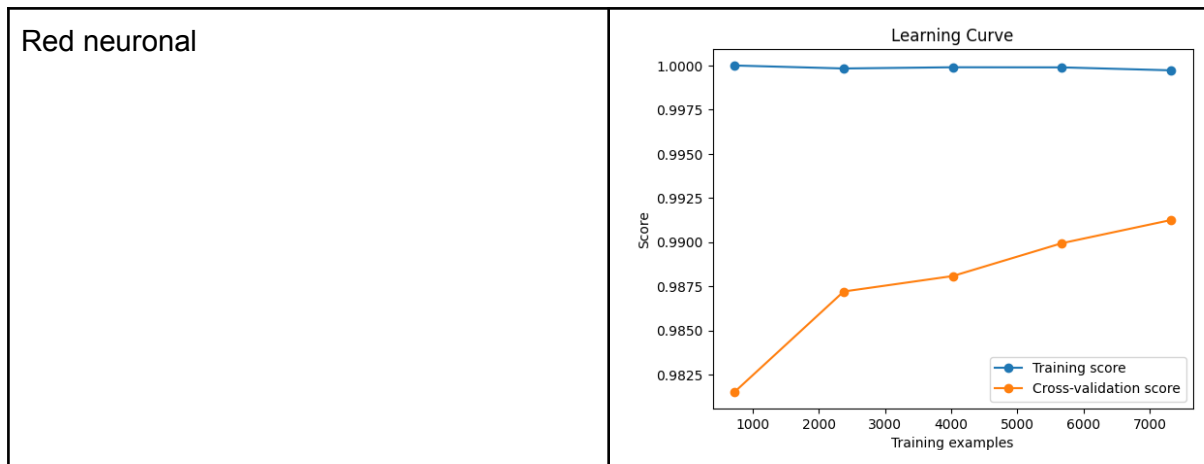


Gradient Boosting



KNN





Elección de modelo final:

El modelo final construido es un clasificador en cascada que comprende a los clasificadores descritos en la sección de “Clasificadores”. El orden que se eligió para la cadena se basó en colocar primero los más simples para rechazar la mayoría de las subventanas antes de recurrir a clasificadores más complejos. Somos conscientes de que no es necesario usar un modelo distinto en cada paso y que podríamos haberlos repetido, pero por los fines académicos de este obligatorio, decidimos cubrir varios de los modelos de aprendizaje vistos en el curso.

El razonamiento que sustenta esta elección es que se pueden descartar rápidamente las subventanas que claramente no tienen un rostro usando los clasificadores simples, mientras que solo se llegan a usar los clasificadores más complejos cuando hay mayores probabilidades de que sea un rostro. Sabiendo que lo más común es que sean pocos los rostros en una imagen, en pocos casos se van a usar todos los clasificadores.

El un principio orden de la cadena fue el siguiente:

1. Regresión logística
2. KNN
3. Bagging
4. Random Forest
5. Gradient Boosting
6. Ada Boost
7. Red neuronal

Se colocaron todos los clasificadores pero hubo varios casos donde algún clasificador decía que una imagen no era un rostro y la quitaba. Para esto, se tuvo que probar distintas combinaciones de los clasificadores para dar con el mejor modelo. Tras varias pruebas agregando, quitando y cambiando órdenes se concluyó que el mejor modelo en cascada comprende a los siguientes clasificadores en orden:

1. Ada Boost
2. Regresión Logística
3. Random Forest
4. Red neuronal

Para construir este clasificador en cascada se creó una clase “CascadeClassifier” donde se definió la función de *predict(X)*. Dicha función tiene la responsabilidad de ejecutar en orden a todos los clasificadores de la cadena y cortar la ejecución si predice que la imagen no contiene un rostro.

En las primeras pruebas que se realizaron para construir este clasificador en cascada se colocó este “filtro” en una función que entrenaba a todos los clasificadores y luego filtraba, pero luego nos dimos cuenta que no era el lugar indicado para realizar dicha acción y lo agregamos a la función de *predict(X)*.

El modelo final y todo el código del proyecto se encuentra en el repositorio mencionado en la carátula de este documento.

Ejemplos de detección de rostros con el modelo final:

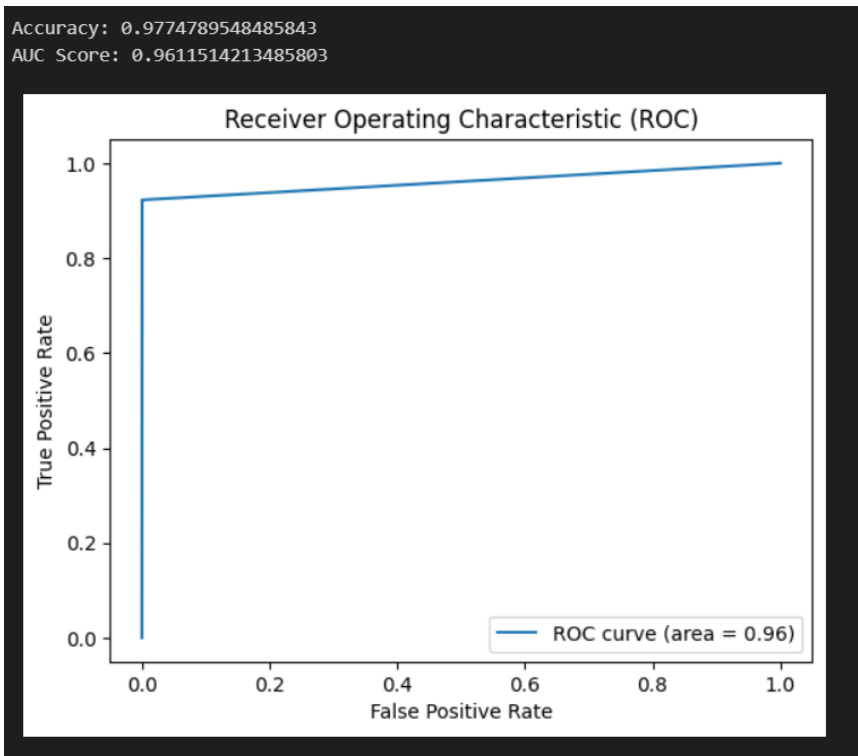


Métricas finales en datos de test:

Accuracy: 0,98

AUC: 0,96

	precision	recall	f1-score	support
0.0	0.97	1.00	0.98	6511
1.0	1.00	0.92	0.96	2636
accuracy			0.98	9147
macro avg	0.98	0.96	0.97	9147
weighted avg	0.98	0.98	0.98	9147



También utilizamos un dataset completamente distinto para verificar los resultados, usando el dataset `sklearn.metrics.olivetti_faces`, siendo los resultados buenos a pesar de lo desbalanceado que era el dataset (algo que perjudicó algunas métricas como el recall).

	precision	recall	f1-score	support
0.0	0.99	1.00	1.00	32500
1.0	0.97	0.27	0.43	400
accuracy			0.99	32900
macro avg	0.98	0.64	0.71	32900
weighted avg	0.99	0.99	0.99	32900

Conclusión:

Este trabajo fue muy útil para integrar muchos conocimientos teóricos adquiridos y demostrar nuestro compromiso al estar al día con los prácticos, lo que sirvió mucho para la parte de código. Tener las soluciones de los prácticos también fue clave ya que lo usamos de respaldo para programar el esqueleto de algunos de los modelos. La parte del proceso que más nos obstaculizó fue importar librerías externas ya que muchas de las usadas en los prácticos dejaron de ser compatibles o cambiaron de nombre, un ejemplo de esto es “pandas-profiling” usado para el preprocesamiento que ahora tiene el nombre “ydata-profiling”.

El modelo que llegamos nos deja muy satisfechos ya que permite reconocer rostros en diferentes ángulos e incluso múltiples en la misma imagen. Siendo el único defecto el encontrado en personas con tonos de piel que no permiten distinguir bien las features que

identificamos en los rostros, algo que en clase se comentó como un problema recurrente en reconocimiento facial incluso en sistemas integrados a bancos. En retrospectiva, quizás podríamos haber aplicado otro preprocesamiento a las imágenes para poder compensar por eso, quizás aumentando el contraste o usando también colores RGB, de todas maneras esto no es un detrimento sustancial ya que el modelo superó nuestras expectativas.

Bibliografía:

- 7.2. *Real world datasets*. (s. f.). scikit-learn.
https://scikit-learn.org/stable/datasets/real_world.html#labeled-faces-in-the-wild-datasets
- *Classifier chain*. (s. f.). scikit-learn.
https://scikit-learn.org/stable/auto_examples/multioutput/plot_classifier_chain_yeast.html
- *LFW Face Database : Main*. (s. f.). <https://vis-www.cs.umass.edu/lfw/>
- Brownlee, J. (2022, 6 agosto). *Use Keras deep learning models with SciKit-Learn in Python*. MachineLearningMastery.com.
<https://machinelearningmastery.com/use-keras-deep-learning-models-scikit-learn-python/>

Además, se usó el material de prácticos de clase e inteligencia artificial generativa como Chat GPT 3.5.