CAMPUS MONTERREY

Inteligencia artificial avanzada para la ciencia de datos I (Gpo 102)

Portafolio de implementación 2

Nombre	Matricula
José Manuel Armendariz Mena	A01197583

Docentes:
Alfredo Esquivel Jaramillo
Mauricio Gonzalez Soto
Frumencio Olivas Alvarez
Antonio Carlos Bento
Hugo Terashima Marín

Septiembre 7 2024

Índice

1. Implementación: Clasificación del marketing de un banco usando Random Forest

Para poder hablar sobre la implementación, es importante primero hablar del dataset, por lo que a continuación se mostrará una descripción a grandes razgos del dataset.

1.1. Dataset

El conjunto de datos se relaciona con campañas de marketing directo (llamadas telefónicas) realizadas por una institución bancaria en Portugal. El objetivo de la clasificación es predecir si un cliente suscribirá un depósito a plazo fijo (variable objetivo: y)[Moro and Cortez, 2014].

1.1.1. Descripción del Conjunto de Datos

El conjunto de datos proporciona información sobre campañas de marketing telefónico llevadas a cabo por un banco en Portugal. A menudo, se requerían múltiples contactos con el mismo cliente para determinar si el producto (un depósito a plazo fijo) sería suscrito o no. El objetivo es predecir si un cliente suscribirá un depósito a plazo basándose en la información recopilada durante la campaña.

1.1.2. Diccionario de Variables

A continuación, se presenta un resumen de las variables contenidas en el conjunto de datos:

Datos del Cliente Bancario:

- **age** (numérica): Edad del cliente.
- **job** (categórica): Tipo de trabajo (e.g., administrador, obrero, emprendedor, etc.).
- **marital** (categórica): Estado civil del cliente (e.g., divorciado, casado, soltero).
- **education** (categórica): Nivel educativo (e.g., primaria, secundaria, universidad, etc.).
- **default** (categórica): ¿Tiene crédito en incumplimiento? (sí/no).
- **housing** (categórica): ¿Tiene préstamo hipotecario? (sí/no).
- **loan** (categórica): ¿Tiene préstamo personal? (sí/no).

Datos del Último Contacto de la Campaña Actual:

- **contact** (categórica): Tipo de contacto (e.g., celular, teléfono fijo).
- **month** (categórica): Mes del último contacto (e.g., enero, febrero, etc.).
- **day_of_week** (categórica): Día de la semana del último contacto (e.g., lunes, martes, etc.).

- **duration** (numérica): Duración del último contacto en segundos.

Otros Atributos:

- **campaign** (numérica): Número de contactos realizados durante la campaña para este cliente (incluye el último contacto).
- **pdays** (numérica): Número de días desde el último contacto en una campaña anterior (999 significa que el cliente no fue contactado previamente).
- **previous** (numérica): Número de contactos realizados antes de esta campaña para este cliente.
- **poutcome** (categórica): Resultado de la campaña de marketing previa (e.g., éxito, fracaso).

Atributos del Contexto Social y Económico:

- **emp.var.rate** (numérica): Tasa de variación del empleo - indicador trimestral.
- **cons.price.idx** (numérica): Índice de precios al consumidor - indicador mensual.
- **cons.conf.idx** (numérica): Índice de confianza del consumidor - indicador mensual.
- **euribor3m** (numérica): Tasa Euribor a 3 meses - indicador diario.
- **nr.employed** (numérica): Número de empleados - indicador trimestral.

Variable Objetivo (a Predecir):

- **y** (binaria): Indica si el cliente suscribió un depósito a plazo (sí/no).

1.1.3. Origen del Conjunto de Datos

El conjunto de datos fue obtenido de la página web del repositorio de aprendizaje automático de la UCI y se puede acceder a través del siguiente enlace:
Fuente: `http://archive.ics.uci.edu/ml/datasets/Bank+Marketing`

1.2. Analisis exploratorio de los datos

Una vez escogido el dataset y determinadas las variables en el dataset, fue necesario explorar y preprocesar el set de datos para mejorar el rendimiento y efectividad del modelo. Lo primero que se hizo fue dividir el dataset en caracteristicas, el cual fueron todas las variables menos y, y etiquetas, el cual fue la variable objetivo y.

1.2.1. Dataset desbalanceado

Durante el análisis exploratorio de los datos, se detectó una desproporción significativa en las etiquetas del dataset. Específicamente, de un total de 41,188 instancias, se encontró que 36,548 registros pertenecían a clientes que no se habían suscrito a un depósito a plazo fijo, mientras que solo 4,640 registros correspondían a clientes que sí lo habían hecho. Esta distribución refleja un claro desbalance en las clases objetivo del dataset.

El desbalance se identificó contando las instancias que decían si o no en la variable objetivo y.

La identificación del desbalance en las etiquetas es crucial, ya que puede afectar negativamente el rendimiento del modelo de machine learning. Los modelos entrenados en datasets desbalanceados tienden a estar sesgados hacia la clase mayoritaria, en este caso, hacia los clientes que no se suscribieron a un depósito a plazo fijo. Esto significa que el modelo podría mostrar un alto rendimiento en general, pero fallar en la correcta clasificación de la clase minoritaria, es decir, los clientes que sí se suscribirían.

2. Preprocesamiento de los datos

Para mitigar los efectos de este desbalance se utilizó la tecnica de Random Under Sampling. El *RandomUnderSampling* consiste en reducir el número de instancias de la clase mayoritaria hasta igualar la cantidad de instancias con la clase minoritaria. Esto se realiza seleccionando aleatoriamente un subconjunto de instancias de la clase mayoritaria y eliminando el resto [Lemaître et al., 2017]. A continuación, se describen los pasos involucrados:

1. **Identificación de las Clases:** Primero, se identifican las clases mayoritaria y minoritaria en el dataset. En el caso del dataset bancario, la clase mayoritaria corresponde a los clientes que no se suscribieron a un depósito a plazo fijo.

2. **Reducción Aleatoria:** Se selecciona aleatoriamente un subconjunto de instancias de la clase mayoritaria, igualando la cantidad de instancias de esta clase con la de la clase minoritaria. Las instancias no seleccionadas se eliminan del dataset.

3. **Nuevo Dataset Balanceado:** El resultado es un nuevo dataset balanceado, donde ambas clases tienen un número similar de instancias. Esto permite que el modelo de machine learning no esté sesgado hacia la clase mayoritaria y pueda aprender mejor a identificar ambas clases.

Ventajas y Desventajas

Las desventajas de Random Under Sampling son:

- **Pérdida de Información:** Al eliminar aleatoriamente instancias de la clase mayoritaria, existe la posibilidad de perder información valiosa, lo que podría afectar el rendimiento general del modelo.

- **Menor Diversidad:** La reducción del tamaño de la clase mayoritaria puede disminuir la diversidad en el dataset, lo que podría limitar la capacidad del modelo para generalizar bien a nuevos datos.

No obstante estas desventajas no son tan significativas debido a que se cuenta con un dataset lo suficientemente grande para poder generalizar de buena manera sin sesgo.

2.1. Transformación de variables numéricas y categoricas

El objetivo del preprocesamiento en este código es aplicar transformaciones específicas a diferentes tipos de características en un conjunto de datos. Se utiliza la clase `ColumnTransformer` de la biblioteca `scikit-learn` para manejar esta tarea.

Definición de Características

Características Numéricas

Las características numéricas son aquellas que contienen datos cuantitativos y se representan con valores numéricos. Las características numéricas se definen como:

caracteristicas_numericas = { .ªge, çampaign, "duration, "pdays, "previous, .emp.var.rate, çons.price.idx, çons.conf.idx, .euribor3m, "nr.employed }

Características Categóricas

Las características categóricas son aquellas que representan categorías o clases y no tienen un orden numérico implícito. Las características categóricas se definen como:

caracteristicas_categoricas = { "job, "marital, .educación, "default, "housing, "loan, çontact, "month, "day_of_week, "poutcome }

Preprocesamiento con `ColumnTransformer`

El objeto `ColumnTransformer` se utiliza para aplicar diferentes transformaciones a diferentes subconjuntos de columnas [Pedregosa et al., 2011a]. En este caso, se definen dos tipos de transformaciones:

- **Transformación para características numéricas**:

$$\text{Transformación numérica} = \text{"passthrough} \tag{1}$$

  Esto significa que las columnas especificadas en `caracteristicas_numericas` se mantienen sin cambios y se pasan tal como están al siguiente paso del procesamiento.

- **Transformación para características categóricas**:

$$\text{Transformación categórica} = \text{OneHotEncoder(handle\_unknown} = \text{ïgnore)} \tag{2}$$

`OneHotEncoder` convierte las características categóricas en una representación binaria. Para cada valor único en una columna categórica, se crea una nueva columna con valores 0 o 1. La opción `handle_unknown=ïgnore"` permite que las categorías que aparecen en los datos de prueba pero que no estaban presentes en los datos de entrenamiento sean ignoradas sin causar errores.

2.2. Creación de pipeline

Se decidió crear un objeto `Pipeline` debido a que este garantiza que las transformaciones aplicadas durante el entrenamiento se mantengan consistentes en la fase de predicción. Esto es esencial para asegurar que los datos de prueba o nuevos datos sean preprocesados de la misma manera que los datos de entrenamiento, evitando inconsistencias que podrían afectar el rendimiento del modelo [Pedregosa et al., 2011b].

Además, un `Pipeline` simplifica la validación cruzada al aplicar automáticamente todos los pasos de preprocesamiento y modelado en cada partición de los datos. Esto reduce el riesgo de errores asociados con el preprocesamiento manual en cada partición, y asegura que el flujo de trabajo sea coherente y fácil de reproducir [Pedregosa et al., 2011b].

Finalmente, el `Pipeline` facilita la optimización de parámetros, permitiendo ajustar simultáneamente los hiperparámetros tanto del preprocesamiento como del modelo en un solo proceso. Esto agiliza la experimentación y mejora el rendimiento del modelo de manera más eficiente [Pedregosa et al., 2011b].

3. Separación y evaluación del modelo en un conjunto de entrenamiento, prueba validación

Se utilizó la función `train_test_split` con un tamaño de prueba de 0.3 y `random_state=42` para dividir el set de datos en prueba y entrenamiento debido a que se deseaba asignar el 30 % de los datos al conjunto de prueba, garantizando al mismo tiempo que la división sea reproducible al establecer una semilla fija para la aleatoriedad. Esto permite realizar experimentos consistentes y en el proceso de modelado.

3.1. Funciones de evaluacion del modelo variando hiperparametros

Se decidieron ajustar los hiperparametros `n_estimators`, `max_depth` y `max_leaf_nodes` debido a que:

- `n_estimators`: Controla el número de árboles en el bosque. Aumentar este valor puede mejorar la precisión del modelo, pero también incrementa el tiempo de entrenamiento.
- `max_depth`: Limita la profundidad máxima de cada árbol. Un valor más alto permite que los árboles capturen más detalles de los datos, pero también puede llevar a sobreajuste.

- `max_leaf_nodes`: Define el número máximo de nodos hoja en cada árbol. Ajustar este parámetro ayuda a controlar la complejidad del modelo y a prevenir el sobreajuste.

Para determinar los rangos en los que se encontrarían valores óptimos, se realizó una evaluación del modelo variando dichos hiperparámetros. Se analizaron métricas clave como la función de pérdida, precisión, sesgo (elevado al cuadrado para facilitar su visualización en gráficos) y varianza. Dado que este proceso es computacionalmente intensivo, se optó por encapsular las evaluaciones del modelo en funciones, permitiendo el uso de procesamiento paralelo para acelerar la ejecución del código.

3.1.1. Función de evaluación del modelo con la metrica: `Accuracy` (precisión)

Primero, se define un `Pipeline` que combina el paso de preprocesamiento de los datos realizado en `ColumnTransformer` con un modelo de Random Forest. Posteriormente, se configura el modelo de Random Forest, donde se ajustaron los hiperparámetros de `n_estimators`, `max_depth` y `max_leaf_nodes`.

A continuación, se utiliza validación cruzada en el conjunto de datos de entrenamiento. Esta técnica divide los datos de entrenamiento en varias particiones, entrenando el modelo en algunas particiones mientras se valida en otras. El proceso se repite varias veces para obtener una estimación robusta de la precisión del modelo en el conjunto de entrenamiento. La precisión media de estas validaciones cruzadas se almacena como una medida del rendimiento del modelo durante el entrenamiento.

Después de la validación cruzada, el `Pipeline` se ajusta a la totalidad del conjunto de datos de entrenamiento. Con el modelo entrenado, se realizan predicciones en el conjunto de datos de prueba. La precisión de estas predicciones se calcula comparando las etiquetas reales con las etiquetas predichas, proporcionando así una medida de rendimiento del modelo en datos no vistos previamente.

Finalmente, la función retorna dos valores clave: la precisión media obtenida durante la validación cruzada en el conjunto de entrenamiento, y la precisión del modelo en el conjunto de prueba. Este proceso se repite variando otros hiperparámetros del modelo para evaluar su impacto en el rendimiento general.

3.1.2. Función de evaluación del modelo con la metrica: `log_loss` (función de perdida)

Primero, se define un `Pipeline` que combina el paso de preprocesamiento de los datos realizado en `ColumnTransformer` con un modelo de `Random Forest`. Posteriormente, se configura el modelo de `Random Forest`, donde se ajustan los hiperparametros al valor proporcionado.

Después, se entrena el `Pipeline` utilizando los datos de entrenamiento. Con el modelo entrenado, se predicen las probabilidades de las clases para los conjuntos de entrenamiento y prueba. Estas probabilidades predichas se utilizan para calcular la métrica `log-loss`, que mide la precisión de las probabilidades predichas

comparándolas con las etiquetas reales.

La `log-loss` se calcula tanto para el conjunto de entrenamiento como para el conjunto de prueba, proporcionando así una medida de rendimiento del modelo en ambos conjuntos de datos. Finalmente, la función retorna dos valores clave: la `log-loss` obtenida durante el entrenamiento y la `log-loss` en el conjunto de prueba. Este proceso se repite variando otros hiperparámetros del modelo, como `max_depth`, `max_leaf_nodes` y `n_estimators` para evaluar su impacto en el rendimiento general.

3.1.3. Función de evaluación del modelo con la metrica: Sesgo, varianza y MSE

Primero, se define un `Pipeline` que combina el paso de preprocesamiento de los datos realizado en `ColumnTransformer` con un modelo de `Random Forest`. Posteriormente, se configura el modelo de `Random Forest`, donde se ajustan los hiperparámetros `max_depth`, `max_leaf_nodes` y `n_estimators` al valor proporcionado.

A continuación, se entrena el `Pipeline` utilizando los datos de entrenamiento. Con el modelo entrenado, se realizan predicciones en los conjuntos de entrenamiento y prueba. Estas predicciones se utilizan para calcular el error absoluto medio (`MAE`), que mide la precisión de las predicciones comparándolas con las etiquetas reales.

Para evaluar la varianza, se calcula el `MAE` de las predicciones en el conjunto de prueba. Una alta varianza indica que el modelo es sensible a las fluctuaciones en los datos de prueba, lo que puede ser un signo de `overfitting` [Kuhn and Johnson, 2013].

El sesgo se evalúa calculando el `MAE` de las predicciones en el conjunto de entrenamiento y elevándolo al cuadrado. Un sesgo alto indica que el modelo no está capturando bien la relación subyacente en los datos de entrenamiento, lo que puede ser un signo de `underfitting` [Kuhn and Johnson, 2013].

Finalmente, se calcula el error cuadrático medio (`MSE`) como la suma de la varianza y el sesgo al cuadrado. Este valor proporciona una medida general del rendimiento del modelo, combinando tanto el error debido al sesgo como el error debido a la varianza [Kuhn and Johnson, 2013].

Este proceso se utiliza para evaluar el impacto de los hiperparametros en el rendimiento general y para identificar posibles problemas de `overfitting` o `underfitting`.

4. Metricas del modelo variando hiperparametros

4.1. Perdida vs numero de estimadores

La figura 1 de `Log-Loss` vs. `Número de Estimadores` muestra cómo el rendimiento del modelo cambia a medida que se ajusta el número de estimadores

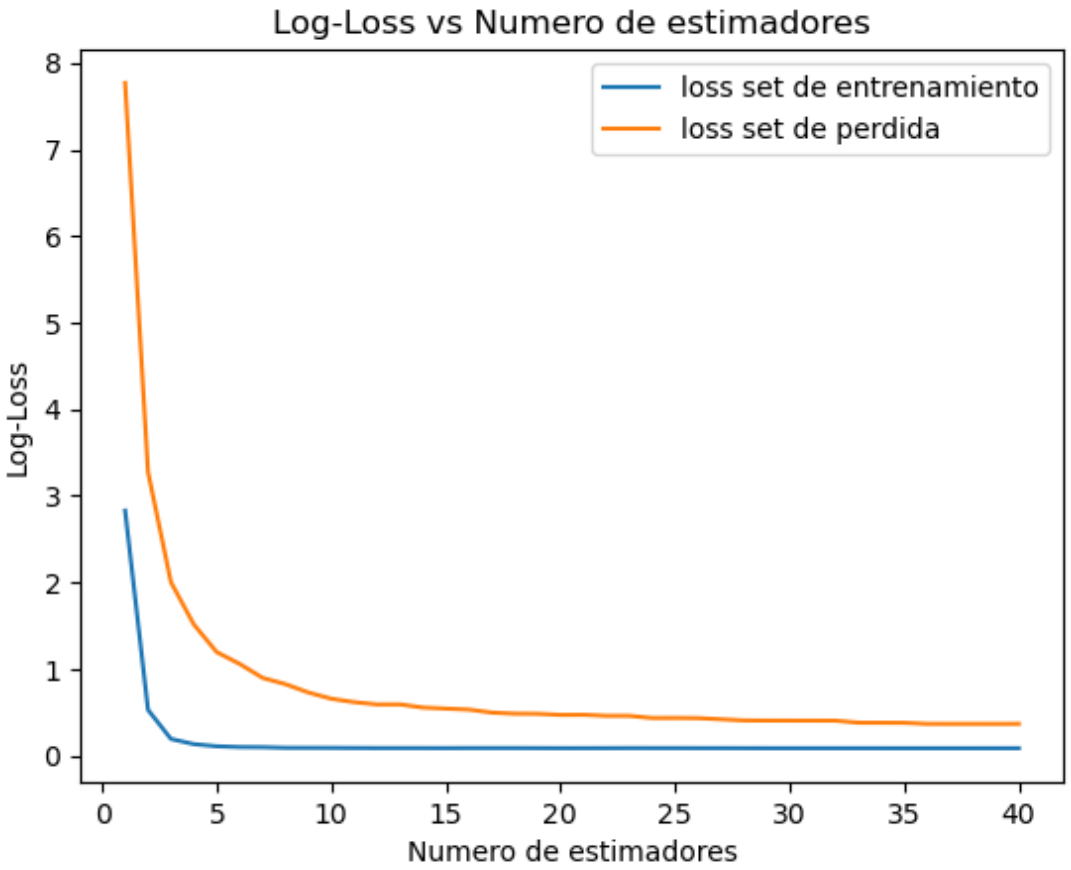

Figura 1: Comportamiento de la perdida del modelo Random Forest al incrementar el numero de estimadores

en un `Random Forest`. Inicialmente, con un número bajo de estimadores, ambas curvas de `log-loss` (entrenamiento y prueba) son altas, indicando que el modelo está `underfitting` y no captura adecuadamente las complejidades de los datos. A medida que se incrementa el número de estimadores, las pérdidas disminuyen rápidamente, lo que sugiere que el modelo está mejorando su capacidad de aprendizaje.

Se observa que las curvas de `log-loss` para el conjunto de entrenamiento y el conjunto de prueba se acercan cada vez más a medida que aumenta el número de estimadores. Esto indica que el modelo está mejorando su capacidad de generalización sin caer en `overfitting`. La disminución continua y paralela de ambas curvas sugiere que el modelo está capturando adecuadamente las complejidades de los datos sin memorizar los datos de entrenamiento.

En conclusión, la figura 1 ilustra claramente la mejora en el rendimiento del modelo a medida que se incrementa el número de estimadores, sin evidenciar signos de `overfitting`. Identificar y mantener el modelo en esta región es crucial para asegurar un buen rendimiento en datos no vistos previamente.

4.2. Perdida vs Profundidad maxima de arbol de decision

La figura 2 de `Log-Loss` vs. `Máxima profundidad` muestra cómo el rendimiento del modelo varía al ajustar la profundidad máxima de los árboles de decisión

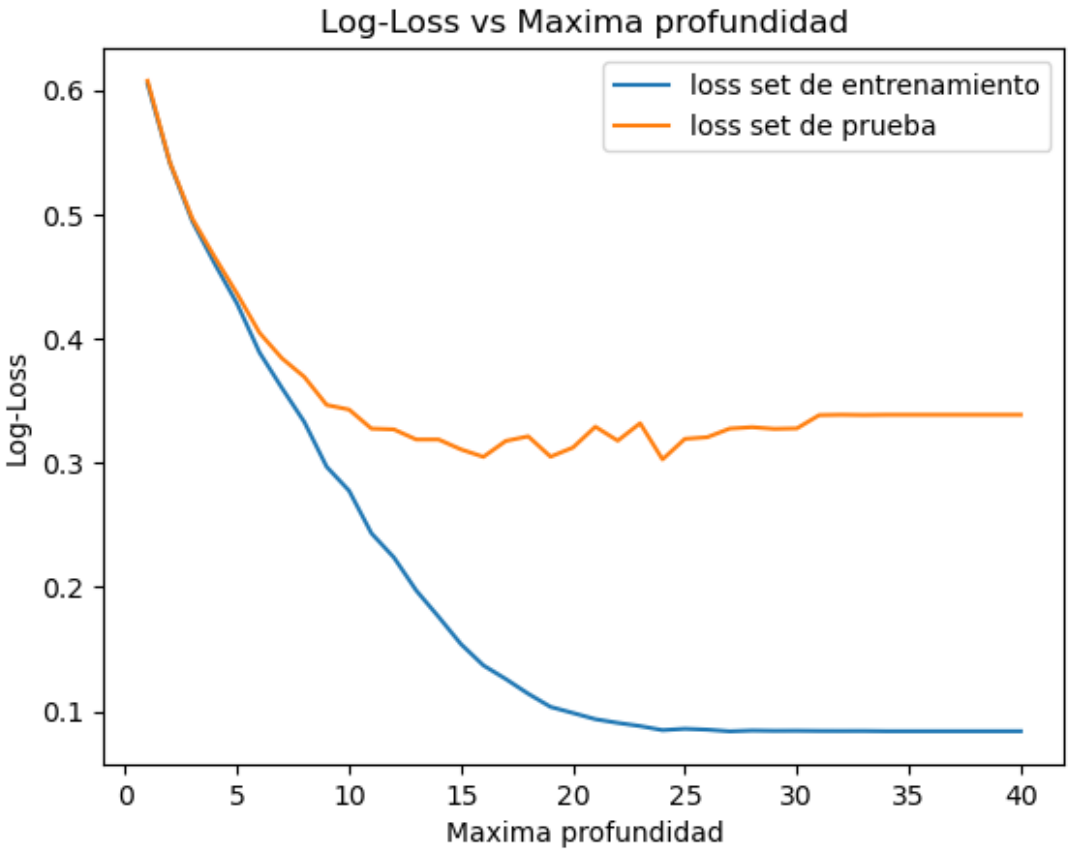

Figura 2: Comportamiento de la perdida del modelo Random Forest al incrementar la profundidad maxima de los arboles

en un `Random Forest`. Con una profundidad baja, ambas curvas de `log-loss` (entrenamiento y prueba) son altas, indicando `underfitting`.

No obstante, a pesar que `log-loss` en el conjunto de prueba disminuye inicialmente y se estabilice esta comienza a fluctuar, incluso incrementando ligeramente. Esto sugiere que el modelo podría estar empezando a `overfitting`, memorizando los datos de entrenamiento sin mejorar en datos nuevos.

En resumen, la figura 2 muestra la transición desde `underfitting` a `overfitting` con el aumento de la profundidad máxima. Es importante identificar la región donde la `log-loss` en el conjunto de prueba es mínima y estable para asegurar un buen rendimiento.

4.3. Perdida vs numero maximo de nodos hoja en cada arbol

En la figura 3 se puede observar que al incrementar el número de nodos hoja en un modelo de `Random Forest`, se observa inicialmente una disminución en el error tanto del conjunto de entrenamiento como del de validación, lo cual indica una mejora en el ajuste del modelo. Sin embargo, a partir de un determinado umbral, el error del conjunto de validación comienza a estabilizarse mientras que la perdida del conjunto de prueba sigue disminuyendo rapidamente, lo cual muestra un posible `overfitting`, para prevenirlo es importante tomar en cuenta que el punto de inflexión en el que se produce este fenómeno marca el numero en el que se tiene buen desempeño y no hay sobreajuste.

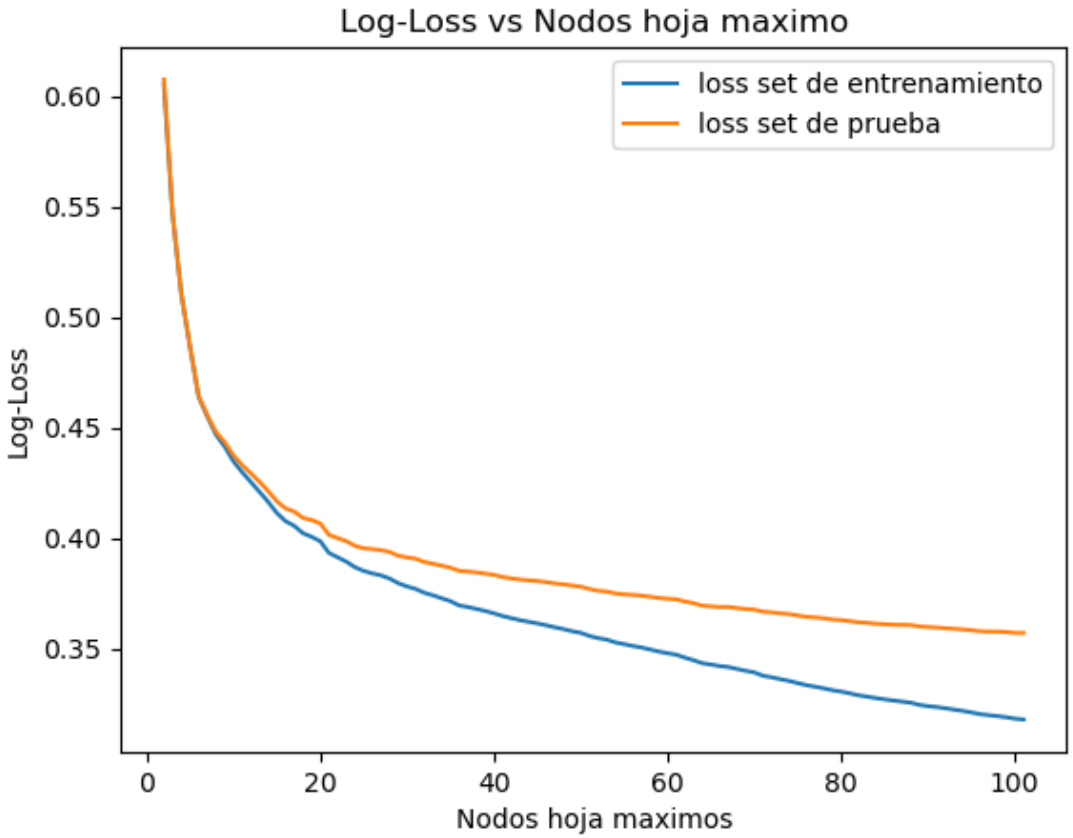

Figura 3: Comportamiento de la perdida del modelo Random Forest al incrementar el numero maximo de los nodos hoja

4.4. Precision vs numero de estimadores

En la figura 4 se observa que la precisión tanto del conjunto de prueba como del conjunto de entrenamiento aumenta a medida que incrementa el número de estimadores, hasta llegar a un punto de estabilización. Este comportamiento indica que inicialmente existe *underfitting*; sin embargo, conforme el modelo se estabiliza, el *underfitting* desaparece. Además, el modelo no muestra signos de *overfitting* al incrementar el número de árboles.

4.5. Precision vs Profundidad maxima de arbol de decision

En la figura 5 se observa que la precisión tanto del conjunto de prueba como del conjunto de entrenamiento aumenta a medida que incrementa la profundidad máxima de los árboles de decisión del modelo, hasta que ambos se estabilizan en un máximo. Al igual que en el caso anterior, no se observa *overfitting*, ya que la precisión del conjunto de prueba no disminuye en ningún punto, ni la precisión del conjunto de entrenamiento sigue aumentando y sobrepasa a la del conjunto de prueba. Por lo tanto, no se observa *overfitting* en esta gráfica.

4.6. Precision vs numero maximo de nodos hoja en cada arbol

En la figura 6, se puede observar la presencia de *underfitting* cuando la cantidad de nodos hoja máximos es pequeña. No obstante, al igual que en los casos

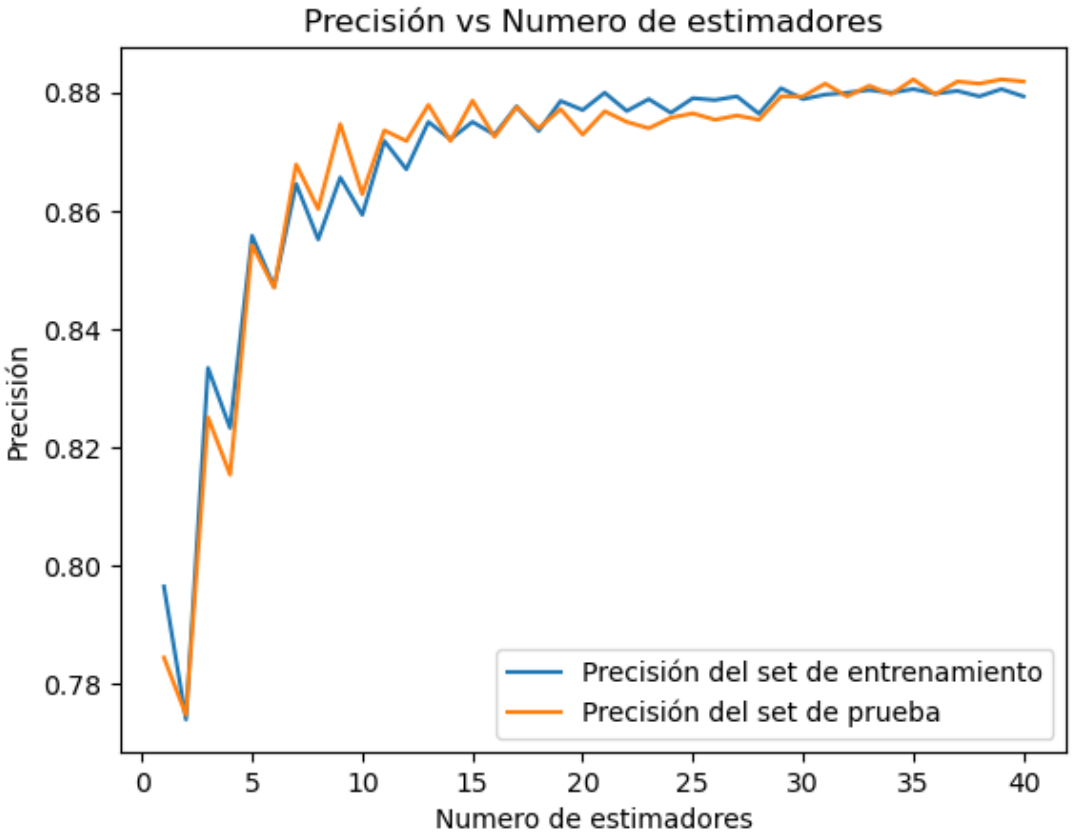

Figura 4: Comportamiento de la precisión del modelo Random Forest al incrementar el numero de estimadores

anteriores, ambas curvas se estabilizan después. En esta gráfica tampoco se observa *overfitting*, por las mismas razones que en la figura 5.

4.7. Sesgo, varianza y MSE vs numero de estimadores

La figura 7 muestra que el modelo no presenta ni *overfitting* ni *underfitting*, ya que el sesgo es bajo y la varianza disminuye a medida que incrementa el número de estimadores. Al mismo tiempo, la varianza del modelo se estabiliza conforme aumenta el número de estimadores. Se considera que el modelo está en su punto ideal cuando el MSE es mínimo. En este caso, el número de estimadores ideal se encuentra entre 100 y 150, ya que en ese rango el modelo está en un punto estable y el MSE es mínimo.

4.8. Sesgo, varianza y MSE vs Profundidad maxima de arbol de decision

La figura 8 muestra un comportamiento muy similar al de la figura 7, con la diferencia de que la varianza tiende a disminuir de manera más dramática y aparentemente se estabiliza más rápido al incrementar la profundidad máxima de los árboles de decisión de Random Forest. Al igual que en el caso anterior, no hay *overfitting* ni *underfitting*, y el rango de profundidad máxima en el cual el modelo se beneficia se encuentra entre 10 y 18.

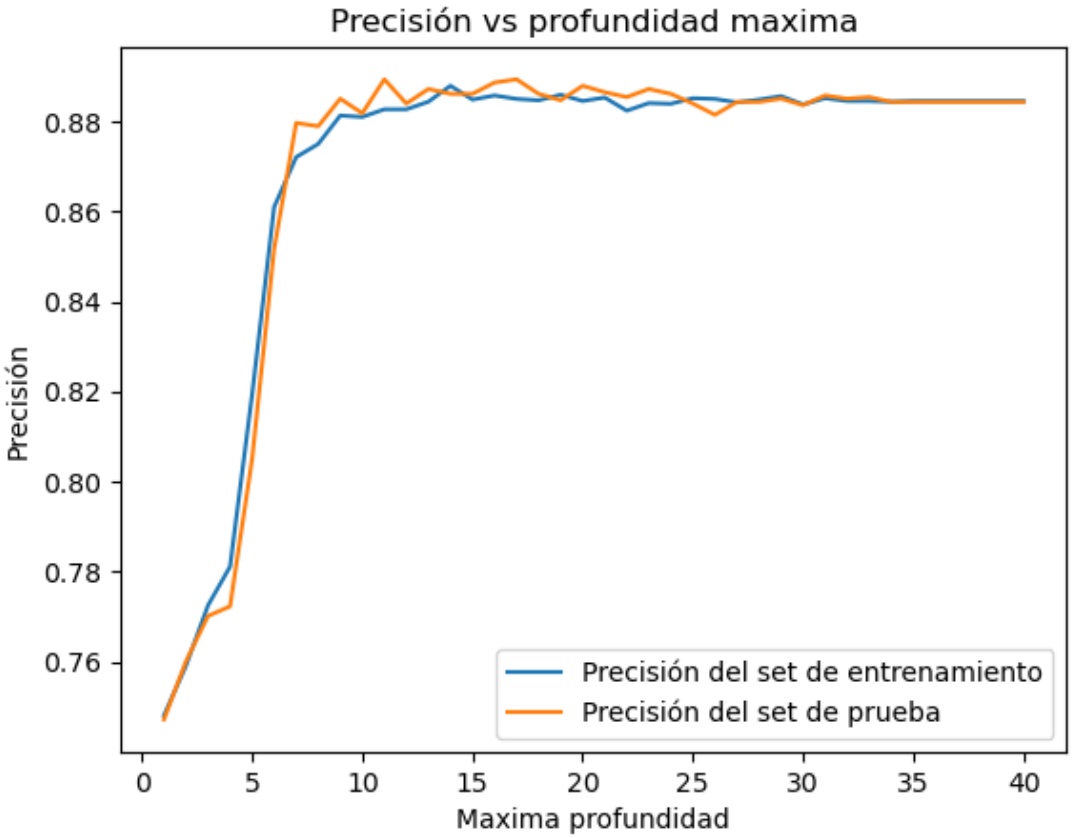

Figura 5: Comportamiento de la precisión del modelo Random Forest al incrementar la profundidad maxima de los arboles

4.9. Sesgo, varianza y MSE vs numero maximo de nodos hoja en cada arbol

Al igual que en los casos anteriores, la figura 9 no muestra *underfitting* ni *overfitting* a medida que aumenta el número máximo de nodos hoja. Se estima que el rango óptimo para este hiperparámetro, para ser considerado un buen modelo, se encuentra entre 35 y 45.

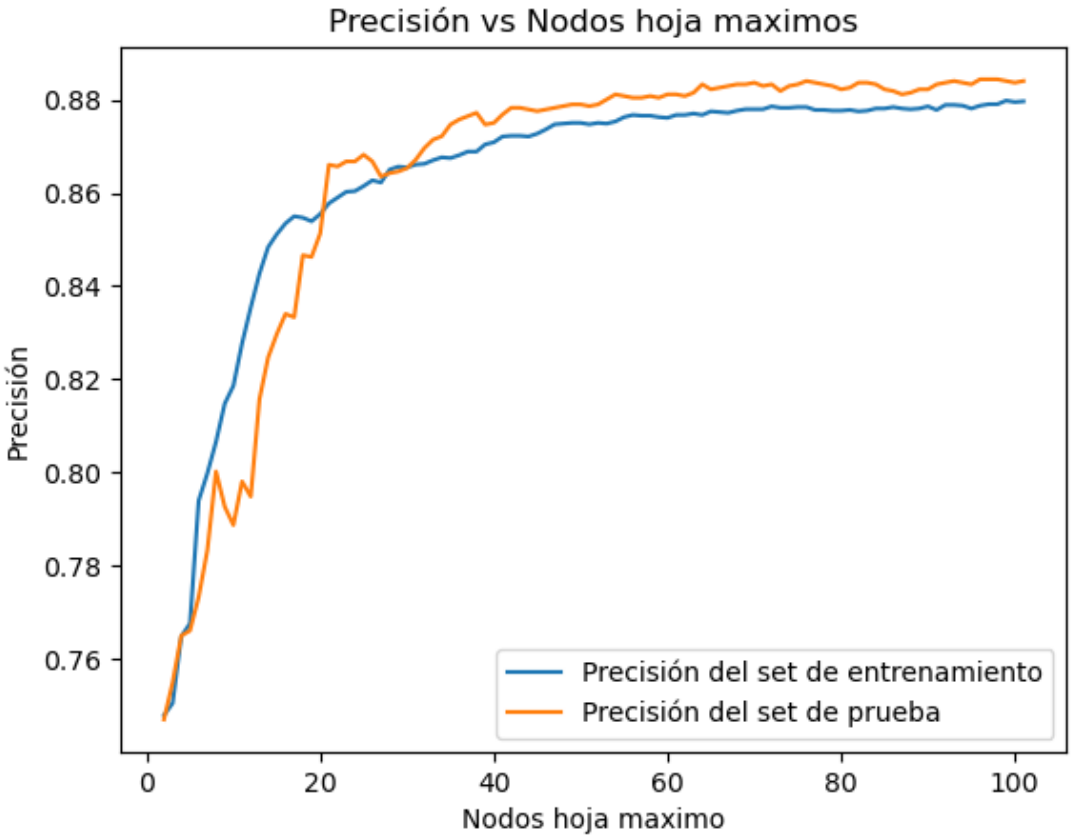

Figura 6: Comportamiento de la precisión del modelo Random Forest al incrementar el numero maximo de los nodos hoja del modelo Random Forest

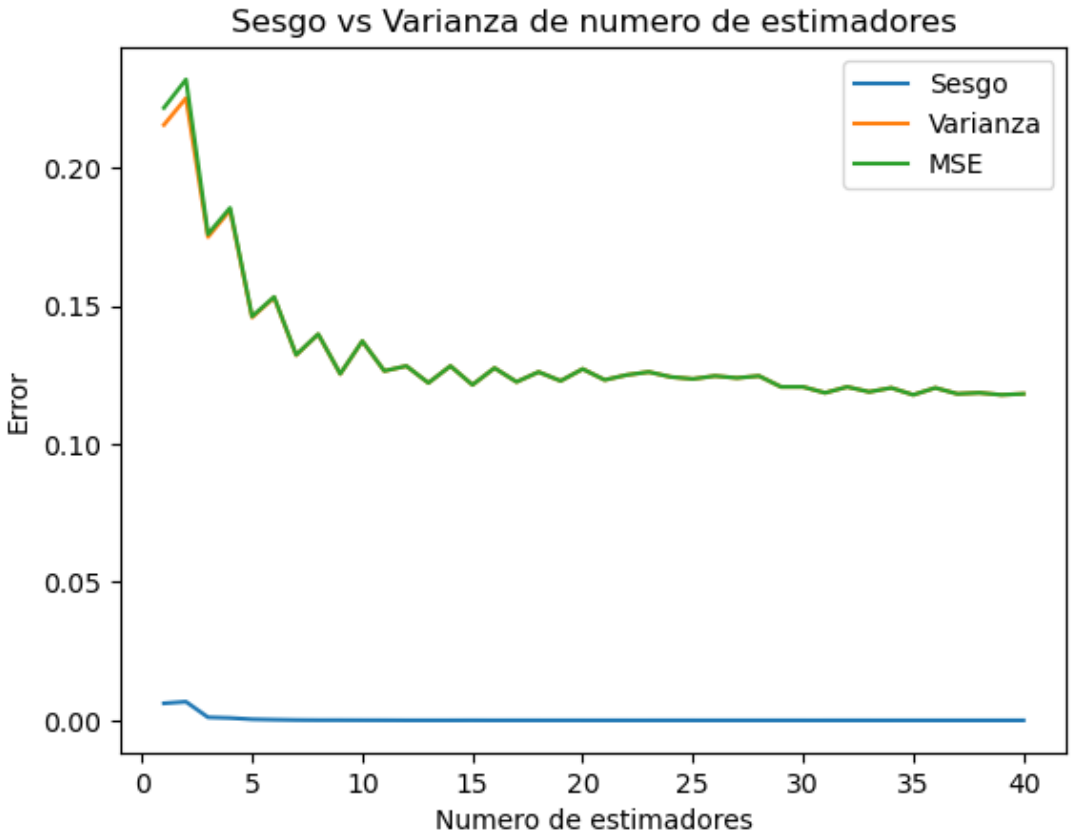

Figura 7: Comportamiento del sesgo, varianza y MSE del modelo Random Forest al incrementar el numero de estimadores

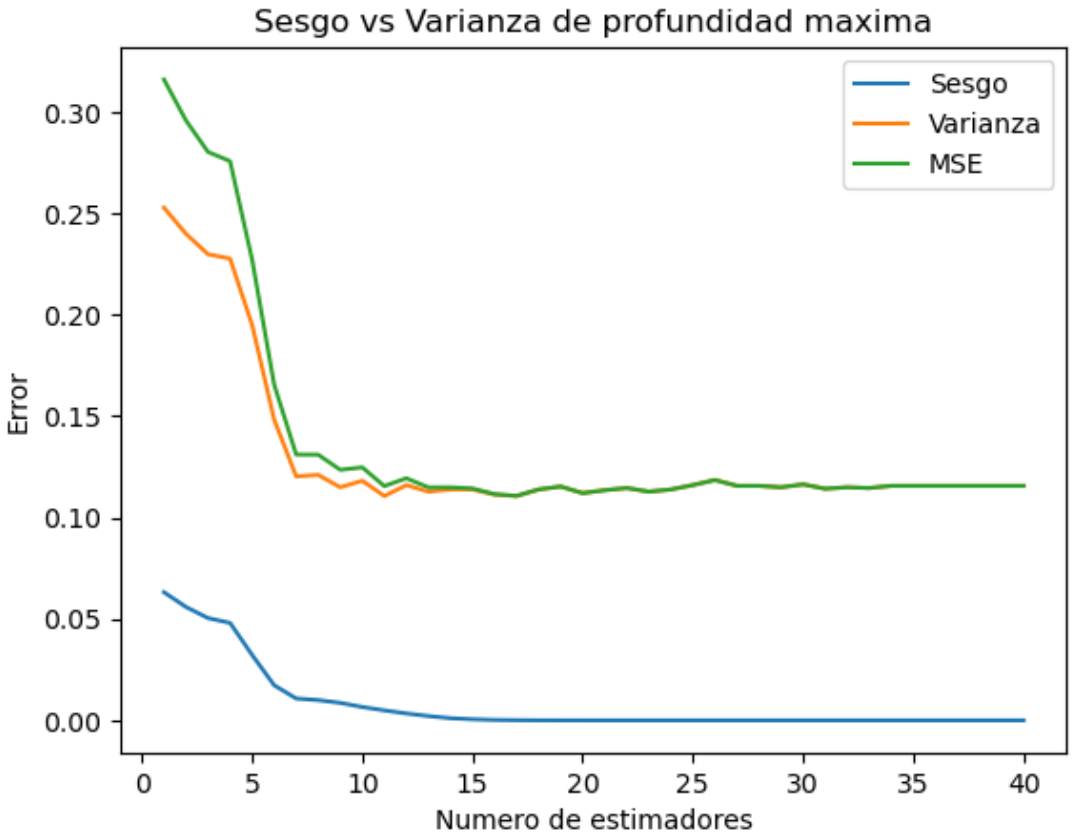

Figura 8: Comportamiento del sesgo, varianza y MSE del modelo Random Forest al incrementar la profundidad maxima de los arboles

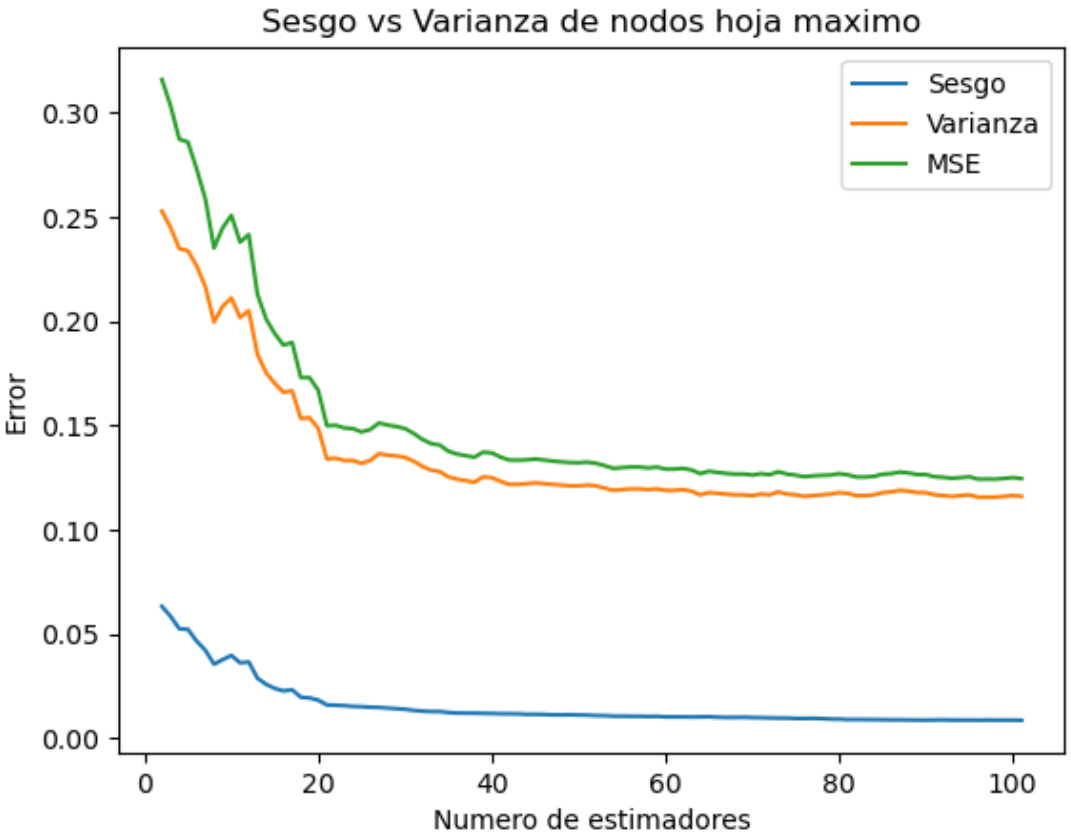

Figura 9: Comportamiento del sesgo, varianza y MSE modelo Random Forest al incrementar el numero maximo de los nodos hoja

5. Ajuste de parametros

En la sección 4 se discutió cómo cambia el desempeño del modelo al variar los hiperparámetros y se encontró un rango aproximado en el cual el modelo generaliza bien sin hacer `overfitting` ni `underfitting`. No obstante, no se conoce el valor exacto de los hiperparámetros ideales.

5.1. Optimización de hiperparametros utilizando `Grid search`

Se utilizó `grid search`, ya que permite realizar una búsqueda exhaustiva sobre un espacio especificado de los hiperparámetros del modelo, evaluando el desempeño de cada combinación posible y seleccionando aquella que optimiza una métrica de interés. En este caso, la métrica seleccionada es la `accuracy`, utilizando una validación cruzada con una división de 5 (`5-fold cross-validation`), ya que esta técnica permite evaluar la estabilidad y la capacidad de generalización del modelo de manera más robusta.

Al dividir los datos en cinco subconjuntos y realizar múltiples iteraciones de entrenamiento y prueba, se obtiene una estimación más precisa del desempeño del modelo en datos no vistos previamente.

Se seleccionó `accuracy` como métrica de interés porque proporciona una medida global del desempeño del modelo, considerando tanto los verdaderos positivos como los verdaderos negativos, considerando que equilibramos nuestro dataset. Si no se hubiera equilibrado el dataset, `recall` hubiera sido mejor metrica ya que se utiliza en contextos donde donde es más importante minimizar los falsos negativos que los falsos positivos, asegurando que el modelo capture la mayor cantidad posible de casos positivos.

Se encontró que los mejores hiperparametros fueron: `max_depth` 18, `leaf_nodes` 45 y `n_estimators` 100

5.2. Criterios de comparación

Se decidió realizar una comparación entre un modelo de `Random Forest` con hiperparametros los siguientes hiperparametros arbitrarios: `max_depth` 2, `leaf_nodes` 2 y `n_estimators` 15 y el modelo con los mejores hiperparametros encontrados para evaluar el desempeño de ambos modelos.

La evaluación del sesgo y la varianza en modelos de `Random Forest` requiere necesariamente la variación de hiperparámetros y muestras debido a la naturaleza misma de estos algoritmos. A diferencia de modelos como las redes neuronales, que se entrenan mediante el descenso de gradiente y permiten una optimización continua de los parámetros del modelo, los `Random Forest` construyen múltiples árboles de decisión de forma aleatoria y combinan sus predicciones. Por esta razón, se decidió utilizar la precisión del conjunto de prueba y del conjunto de entrenamiento (para corroborar que no hay ni *overfitting* ni *underfitting*) y matrices de confusión.

5.3. Comparación de resultados

5.3.1. Resultados del modelo de hiperparametros arbitrarios

El modelo mostró una precisión en el conjunto de prueba de 0.7385 y en el conjunto de entrenamiento del 0.7421. Esto muestra que el modelo con los parámetros arbitrarios es bueno generalizando y que no hace ni `overfitting` ni `underfitting`.

En la figura 10 se mostrará la matriz de confusión del modelo con hiperparámetros arbitrarios.

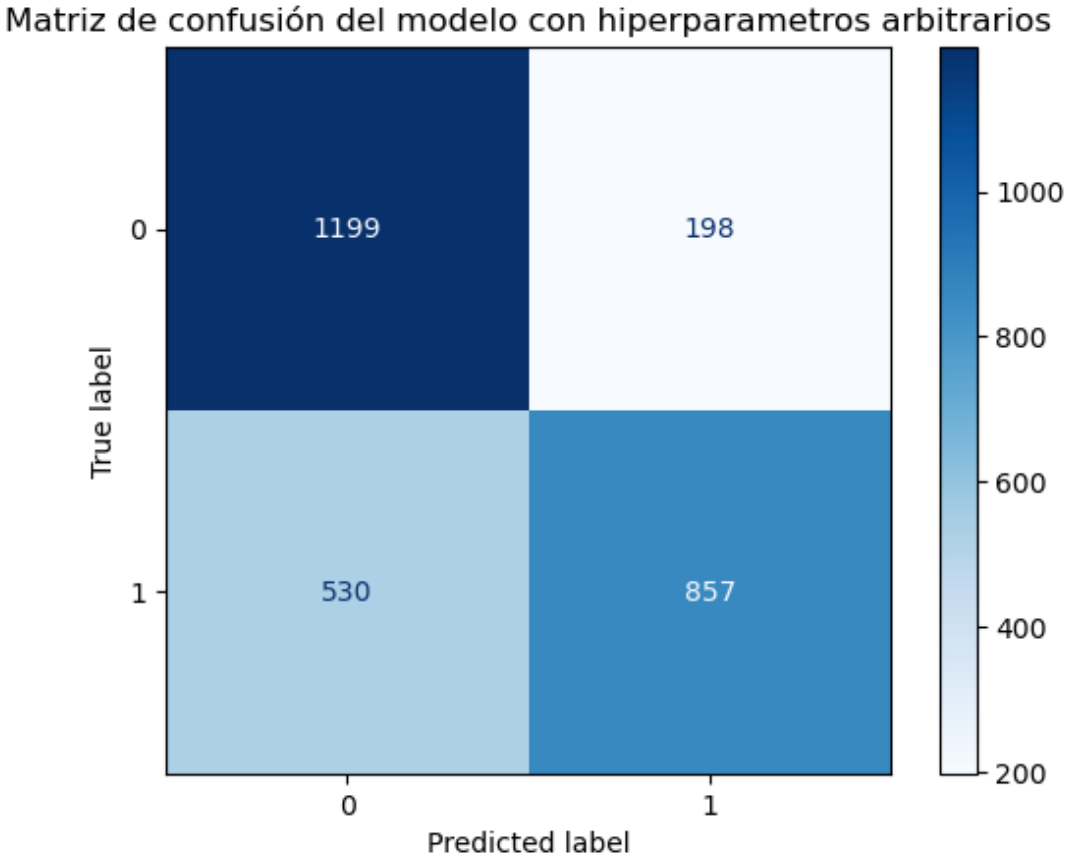

Figura 10: Matriz de confusión del modelo `Random Forest` con hiperparametros arbitrarios

Como se puede ver en la figura 10 a pesar que la precisión fue buena, la matriz de confusión muestra un error grande ya que hay muchos falsos negativos, lo que significa que el modelo tiene un sesgo alto hacia los falsos negativos.

5.3.2. Resultados del modelo de hiperparametros optimizados

El modelo mostró una precisión en el conjunto de prueba de 0.8681 y en el conjunto de entrenamiento del 0.8814. Esto muestra que igual que con el modelo con los parámetros arbitrarios, el modelo con hiperparametros optimizados es bueno generalizando y que no hace ni `overfitting` ni `underfitting`.

En la figura 11 se mostrará la matriz de confusión del modelo con hiperparámetros optimizados.

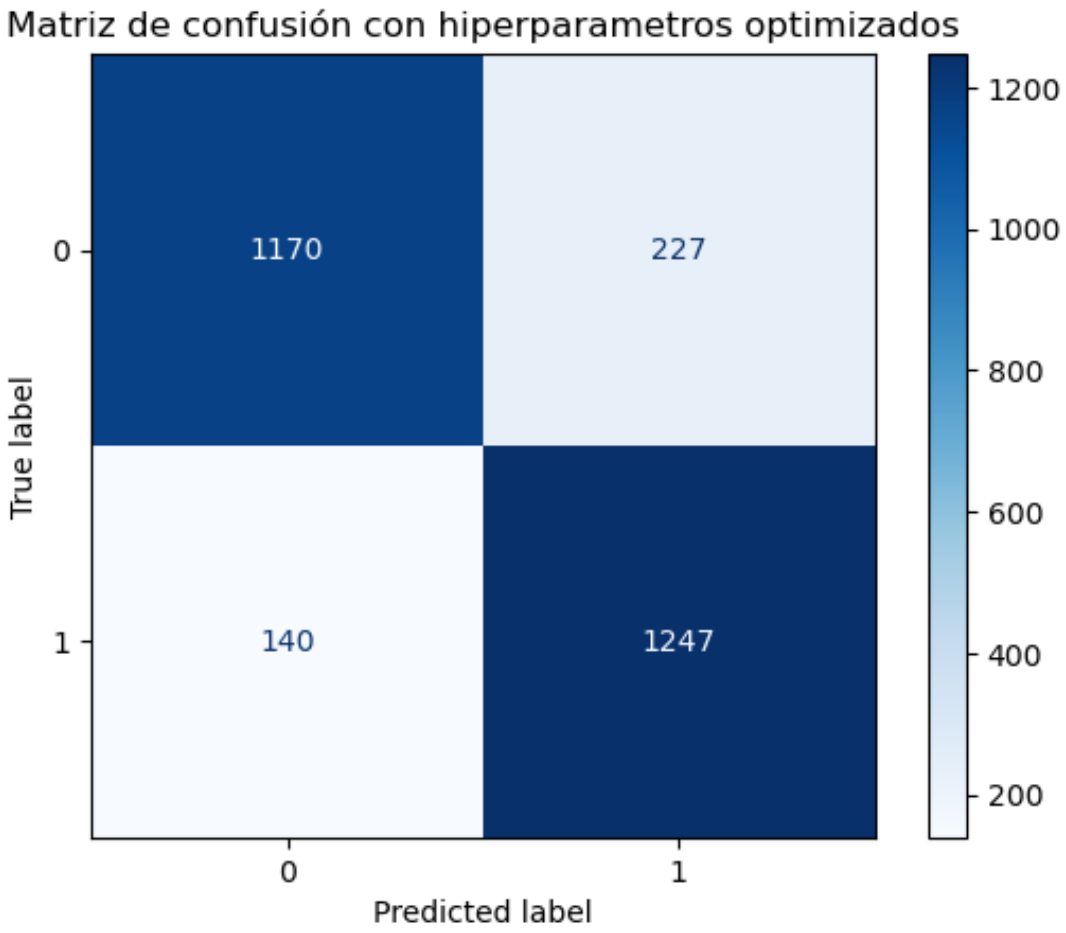

Figura 11: Matriz de confusión del modelo `Random Forest` con hiperparametros optimizados

Como se puede ver en la figura 11 no solo la precisión del modelo fue buena, sino que también es capaz de predecir los verdaderos positivos y verdaderos negativos con buena precisión, esto significa que el modelo es bueno generalizando y que no esta sesgado.

5.3.3. Comparación general

Si bien, ambos modelos se desempeñaron bien, el modelo con los hiperparametros optimizados fue mejor tanto en la generalización del modelo, como en la predicción de positivos y negativos verdaderos, lo cual lo hace un mejor modelo que el que tiene los hiperparametros arbitrarios.

6. Conclusión

Los modelos de `Random Forest` son conocidos por su capacidad para tener un sesgo bajo debido a su naturaleza de ensamblaje, donde múltiples árboles de decisión se combinan para realizar predicciones. Esta característica permite que el modelo capture patrones complejos en los datos, lo que resulta en una menor probabilidad de cometer errores sistemáticos en la clasificación. Sin embargo, esta misma capacidad de modelar complejidades también los hace susceptibles a una alta varianza. Esto ocurre porque, a medida que se añaden más árboles, cada

árbol puede aprender diferentes aspectos de los datos de entrenamiento, lo que puede llevar a que el modelo se ajuste demasiado a las particularidades de esos datos, en lugar de generalizar bien a nuevos datos.

En el caso de nuestro modelo, se logró evitar el `overfitting` a pesar de la complejidad inherente a los modelos de `Random Forest`. Esto se debió a varios factores. Primero, se utilizó un conjunto de datos balanceado, lo que ayudó a que el modelo no se sesgara hacia la clase mayoritaria, permitiéndole aprender de manera más efectiva sobre ambas clases. Además, el rango de exploración de los hiperparametros para observar el comportamiento de la perdida, precisión, sesgo y varianza no fue lo suficientemente alto como para crear `overfitting`.

Finalmente, se encontraron valores óptimos para los hiperparámetros dentro de un rango que evita el `overfitting`, asegurando que el modelo mantenga un buen rendimiento tanto en el conjunto de entrenamiento como en el de prueba. Por lo tanto, la combinación de un enfoque cuidadoso en el preprocesamiento de datos, la optimización de hiperparámetros y el uso de técnicas de validación robustas fueron fundamentales para que el modelo `Random Forest` no presentara signos de `overfitting` y `underfitting` y obtuviera una precisión alta.

7. Referencias

Referencias

[Kuhn and Johnson, 2013] Kuhn, M. and Johnson, K. (2013). *Applied predictive modeling*. Springer.

[Lemaître et al., 2017] Lemaître, G., Nogueira, F., and Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5. The RandomUnderSampler is a part of the imbalanced-learn library, which is built on top of scikit-learn.

[Moro and Cortez, 2014] Moro, S., R. P. and Cortez, P. (2014). Bank Marketing. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5K306.

[Pedregosa et al., 2011a] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, (2011a). scikit-learn: ColumnTransformer Documentation. `https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html`.

[Pedregosa et al., 2011b] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, (2011b). scikit-learn: Pipeline Documentation. `https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html`.