Master's degree in Statistics and Operations Research
Statistical Learning

Task 1: Prediction with Trees and Ensemble-based trees

Anna Aguilera

Gaspar Lloret

Barcelona, Spain
13 de marzo de 2025

Índice

1. Introduction

En este trabajo se presenta la implementación, la evaluación y la optimización de hiperparámetros de cuatro modelos de clasificación diferentes basados en árboles de predicción. Los modelos implementados son *Classification and Regression Trees* (CART), *Random Forest* (RF), *XGBoost*, y *Bayesian Trees*.

La estructura del informe incluye una descripción detallada de cada modelo y las observaciones pertinentes sobre su implementación. También se adjunta el código desarrollado, acompañado de comentarios exhaustivos que explican cada paso y decisión tomada en el proceso de programación. Como complemento, se presenta una comparación de la precisión de cada modelo.

Los árboles de predicción son modelos estructurados jerárquicamente que dividen el espacio de los datos en splits mediante reglas binarias. Cada nodo del árbol representa una decisión basada en una variable, mientras que las hojas contienen las predicciones finales. Su facilidad de interpretación y adaptabilidad los convierte en una buena opción a la hora de modelar predicciones, aunque pueden ser propensos al sobreajuste, limitando su capacidad para generalizar en datos nuevos. Para superar estas limitaciones,los métodos de ensamble combinan múltiples modelos para mejorar la precisión y robustez. Existen dos enfoques principales: **Bagging**, entrena varios árboles en paralelo con diferentes subconjuntos de datos y combina sus predicciones, como en Random Forest; y **Boosting**, el cual construye árboles secuencialmente, donde cada nuevo árbol corrige los errores del anterior, como en Gradient Boosting.

Este informe busca cumplir con los requerimientos de la práctica, proporcionando una documentación detallada y comprensible del código, junto con la comparación de los modelos.

2. Descripción de los datos

El dataset contiene información sobre diferentes factores relacionados con el crecimiento de 697 flores. Estos factores son el tiempo que tardan en crecer (en días), el tamaño (en centímetros), el área que cubren (en cm^2) y 19 variables que corresponden a genotipos de distintos genes.

Una exploración inicial nos permite observar como el desbalance de clases (Figura 2.1), no es muy grande (38 % vs 62 %).

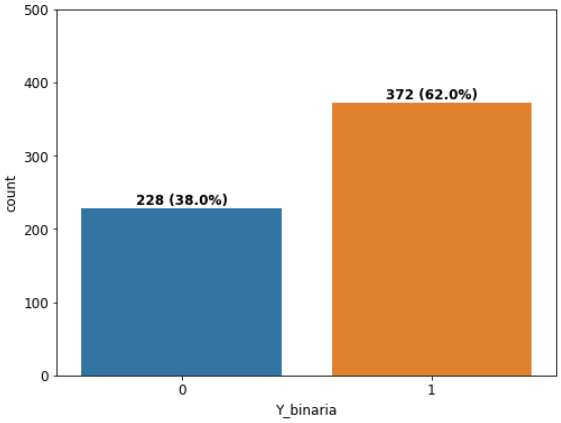

Figura 2.1: Proporción de clases

En estos datos no encontramos ningún missing, por lo tanto podemos trabajar directamente con ellos. Además, en la figura 2.2, podemos observar que las únicas 2 variables contínuas (longitud y área) no están correlacionadas.

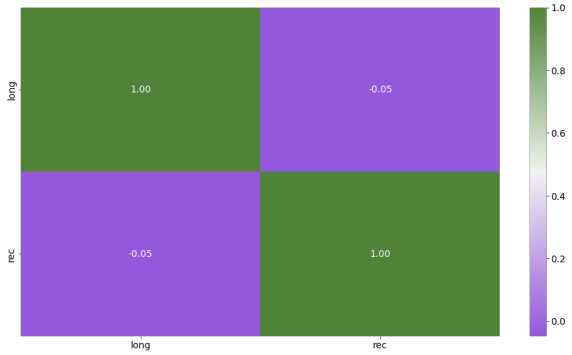

Figura 2.2: Enter Caption

3. Decision Trees (CART)

El algoritmo de los árboles de decisión funciona mediante la división de los datos en distintos grupos basándose en el atributo más significante en cada nodo. Este proceso continua recursivamente hasta que se alcanza un nodo hoja o todos los atributos se han utilizado. El objetivo es crear un modelo que predice el valor de la variable de interés mediante el aprendizaje de normas de decisión inferidas de los datos.

Para implementar este modelo en Python, utilizamos la función tree.DecisionTreeClassifier del paquete scikit-learn. Los hiperparámetros que forman este tipo de modelos son:

- Criterion: mide la calidad del split. Puede ser Gini impurity o Shannon information gain.
- Splitter: escoger el mejor split o el mejor split aleatorio.
- max_depth: profundidad máxima.
- min_samples_split: Número mínimo de muestras para generar un split.
- min_samples_leaf: número mínimo de muestras en un nodo hoja.
- min_weight_fraction_leaf: pesos para los nodos hoja.
- max_features: número de predictores cuando se busca el mejor split.
- class_weight: pesos balanceados o no.

Elegir bien estos parámetros es fundamental para dar con el modelo más óptimo según los datos con los que trabajamos. Para este modelo hemos generado un modelo con los parámetros por defecto y otro a partir de los parámetros que mejor se ajustan según los resultados del Grid Search con 5 Cross Validaciones. La ventaja de generar un modelo a partir de los hiperparámetros por defecto, es el ahorro computacional de llevar a cabo el tuning de hyperparámetros. Como podemos ver resumido en la Tabla 3.1, en este caso no cambian muchos de los hiperparámetros, esto es debido a que los hiperparámtros por defecto están preparados de la manera más genérica posible.

Parámetro	Por Defecto	Optimizado
criterion	gini	entropy
splitter	best	best
max_depth	None	4
min_samples_split	2	2
min_samples_leaf	1	1
min_weight_fraction_leaf	0.0	0.0
max_features	None	None
random_state	None	None
max_leaf_nodes	None	None
min_impurity_decrease	0.0	0.0
class_weight	None	None
ccp_alpha	0.0	0.0
monotonic_cst	None	None

Tabla 3.1: Parámetros por defecto y optimizados

3.1. Evaluación

Después de construir los modelos, tenemos que evaluar su capacidad predictiva. Las métricas que hemos utilizado son la tasa de aciertos, también se puede ver en la matriz de confusión (Figura 3.1), la sensibilidad y especificidad (Tabla 3.2), y la curva ROC que resume todas las métricas (Figura 3.2).

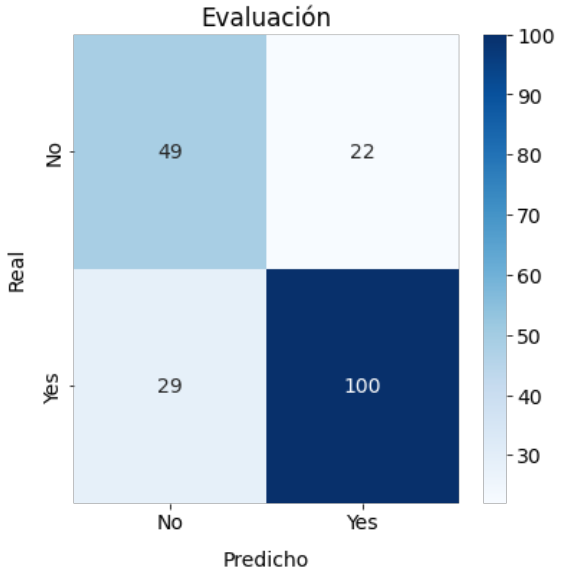

Figura 3.1: Matriz de confusión, Evaluación

Métricas	Por Defecto	Optimizado
Precisión	0.74	0.85
Sensibilidad	0.78	0.92
Especificidad	0.69	0.72
AUC	0.73	0.82

Tabla 3.2: Parámetros por defecto y optimizados

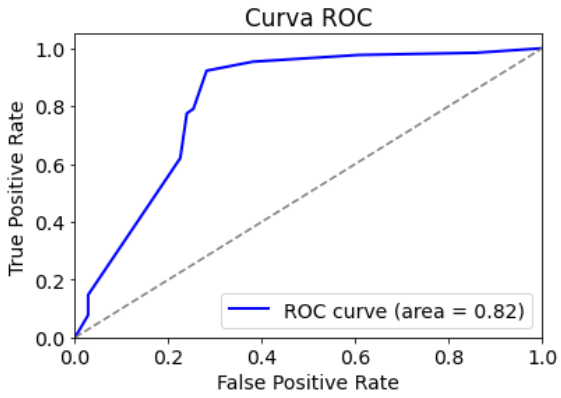

Figura 3.2: Curvas Roc para modelos CART

Estos resultados muestran como la optimización de hiperparámetros mejora los resultados de todas las métricas. Al ajustar adecuadamente los hiperparámetros, el modelo puede lograr una mayor precisión, reducir el error y mejorar su capacidad de generalización. Esto se traduce en predicciones más confiables y en una mejor adaptación a datos no vistos. Además, la optimización permite encontrar un equilibrio adecuado entre sesgo y varianza, evitando problemas como el sobreajuste o el subajuste.

4. Random Forest (RF)

El algoritmo de Random Forest funciona de manera similar al de árboles de decisión, ya que ambos construyen modelos basados en estructuras jerárquicas de reglas de decisión. Sin embargo, la diferencia clave radica en que Random Forest no se basa en un único árbol, sino en un conjunto de múltiples árboles de decisión. Mientras que un árbol de decisión toma decisiones a partir de un único conjunto de reglas derivado de los datos de entrenamiento, Random Forest combina varios árboles, cada uno entrenado con una muestra diferente del conjunto de datos (mediante bagging) y con una selección aleatoria de características en cada división.

Los hiperparámetros que podemos editar en este tipo de modelos son los siguientes:

- n_estimators: número de árboles en el Forest.
- Criterion: mide la calidad del split. Puede ser Gini impurity o Shannon information gain.
- max_depth: profundidad máxima.
- min_samples_split: Número mínimo de muestras para generar un split.
- min_samples_leaf: número mínimo de muestras en un nodo hoja.
- max_leaf_nodes: Número máximo de nodos hoja.
- bootstrap: si se utiliza esta técnica o no.
- oob_score: si se utilizan muestras out-of-bag para estimar el error de generalización.

- n_jobs: el número máximo de tareas en paralelo.

- max_features: número de predictores cuando se busca el mejor split.

Para este modelo hemos generado también un modelo con los parámetros por defecto y otro a partir de los parámetros que mejor se ajustan según los resultados del Grid Search con 5 Cross Validaciones. La ventaja de generar un modelo a partir de los hiperparámetros por defecto, es el ahorro computacional de llevar a cabo el tuning de hyperparámetros. Como podemos ver resumido en la Tabla 4.1, en este caso no cambian muchos de los hiperparámetros, esto es debido a que los hiperparámtros por defecto están preparados de la manera más genérica posible.

Parámetro	**Por Defecto**	**Optimizado**
criterion	gini	entropy
n_estimators	10	50
max_depth	None	None
min_samples_split	2	2
min_samples_leaf	1	1
max_features	auto	auto
bootstrap	True	True
max_leaf_nodes	None	None
oob_score	False	False

Tabla 4.1: Parámetros por defecto y optimizados

El gráfico de a continuación (Figura 4.1), muestra como varia el score según la combinación de hiperparámetros seleccionada. En este caso, el mejor score lo obtenemos con 50 estimadores, criterio entropy y máxima profundidad de 6.

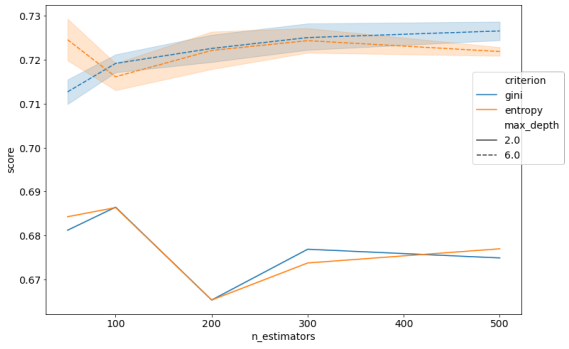

Figura 4.1: Optimización de hiperparámetros

4.1. Evaluación

Como en el caso del modelo CART, hemos llevado a cabo una evaluación del poder predictivo de los modelos entrenados con nuestros datos. Las métricas son las mismas que en el caso anterior, en la Figura 4.2 encontramos la precisión o tasa de aciertos en forma de matriz de confusión, la sensibilidad y especificidad (Tabla 4.2), y la curva ROC que resume todas las métricas (Figura 4.3).

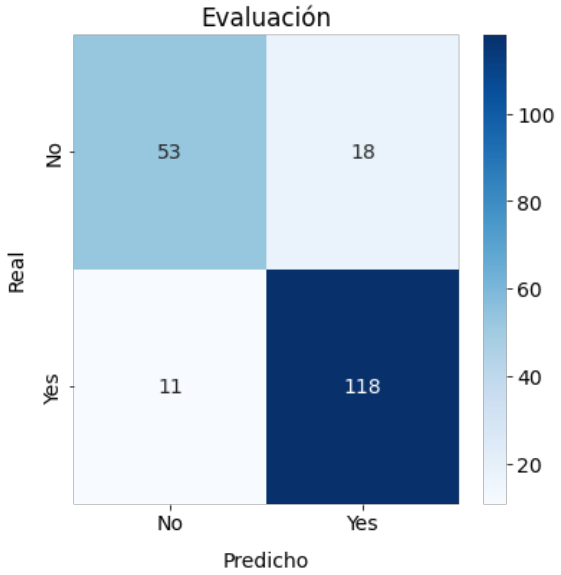

Figura 4.2: Matriz de confusión, Evaluación

Métricas	**Por Defecto**	**Optimizado**
Precisión	0.85	0.83
Sensibilidad	0.91	0.90
Especificidad	0.75	0.70
AUC	0.91	0.90

Tabla 4.2: Parámetros por defecto y optimizados

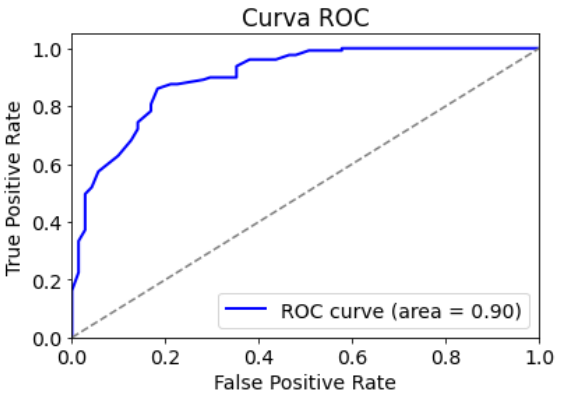

Figura 4.3: Curvas ROC

Estos resultados muestran como la optimización de hiperparámetros no consigue mejorar los resultados de todas las métricas, aunque con una diferencia muy pequeña. Esto puede ser debido a que este modelo está sobreajustado a los datos de entrenamiento, lo que empeora sus resultados en la generalización. Al ajustar los hiperparámetros, el modelo puede lograr una mayor precisión y reducir el error en la partición de entrenamiento, pero no capturar la realidad de la partición de test.

5. Extreme Gradient Boosting (XGBoost)

XGBoost es un algoritmo de aprendizaje automático supervisado basado en el framework de gradient boosting. Es un método de aprendizaje en conjunto que construye múltiples árboles de decisión de manera secuencial, donde cada árbol corrige los errores del anterior. A diferencia de las técnicas tradicionales de boosting, XGBoost incorpora regularización mediante penalizaciones L1 y L2, poda de árboles y paralelización, lo que lo hace altamente eficiente, escalable y resistente al sobreajuste. El algoritmo optimiza una función objetivo utilizando gradientes de segundo orden, lo que permite una mejor aproximación y una convergencia más rápida. XGBoost se utiliza ampliamente para tareas de clasificación y regresión, siendo especialmente popular en competiciones de machine learning y aplicaciones en áreas como finanzas, salud y sistemas de recomendación.

Estos son algunos de los hiperparámetros que podemos editar en este modelo:

- **n_estimators**: número de árboles en el modelo. Un valor mayor puede mejorar el rendimiento, pero aumenta el tiempo de entrenamiento. (Default: 100)
- **max_depth**: profundidad máxima de los árboles. Un valor alto permite capturar más relaciones en los datos, pero puede llevar a sobreajuste. (Default: 6)
- **learning_rate**: tasa de aprendizaje que controla cuánto contribuye cada árbol a la predicción final. Un valor más bajo mejora la generalización pero requiere más árboles. (Default: 0.3)
- **subsample**: fracción del conjunto de datos utilizada en cada árbol. Reduce el sobreajuste al entrenar cada árbol con una muestra aleatoria. (Default: 1)
- **colsample_bytree**: fracción de predictores seleccionados en cada árbol, lo que ayuda a reducir la correlación entre árboles y mejorar la generalización. (Default: 1)
- **gamma**: reducción mínima de la función de pérdida requerida para hacer un split. Un valor mayor evita divisiones innecesarias y reduce el sobreajuste. (Default: 0)
- **use_label_encoder**: si se usa el codificador interno de etiquetas de XGBoost. Se establece en `False` para evitar advertencias en versiones recientes.
- **eval_metric**: métrica de evaluación utilizada para optimizar el modelo. En este caso, se usa `logloss`, que mide la entropía cruzada para problemas de clasificación.

A continuación, se presentan los hiperparámetros utilizados en el modelo XGBoost. Se ha llevado a cabo un proceso de ajuste probando diferentes configuraciones para encontrar la combinación que optimiza el rendimiento del modelo. Se ha incrementado el número de árboles (`n_estimators`) para mejorar la capacidad de aprendizaje, y se ha ajustado la tasa de aprendizaje (`learning_rate`) para lograr una mejor convergencia. Además, se ha reducido el muestreo de datos (`subsample`) y de características (`colsample_bytree`) para evitar sobreajuste. La métrica de evaluación utilizada es `logloss`, ya que el problema es de clasificación binaria y esta métrica mide la entropía cruzada, evaluando la calidad de las probabilidades generadas por el modelo.

Parámetro	**Por Defecto**	**Optimizado**
n_estimators	100	2000
max_depth	6	10
learning_rate	0.3	0.9
subsample	1	0.8
colsample_bytree	1	0.8
gamma	0	0.1
use_label_encoder	False	False
eval_metric	-	logloss

Tabla 5.1: Parámetros por defecto y optimizados

5.1. Evaluación

Seguidamente se presenta la Confusion Matrix para una representación más visual, juntamente con una tabla con las métricas evaluadoras del modelo entrenado y un gráfico de la curva ROC:

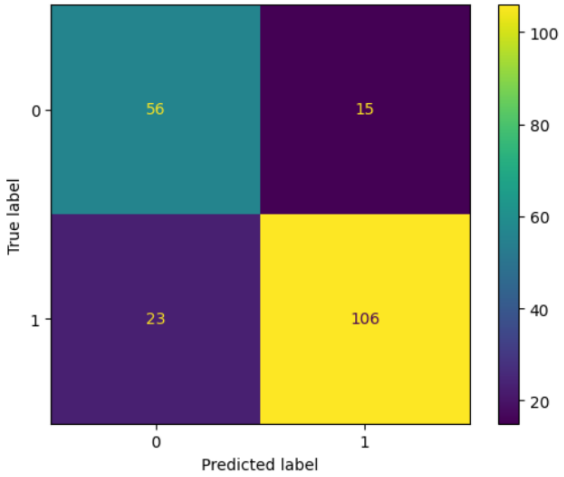

Figura 5.1: Matriz de confusión del XGBoost

Métricas	Resultado
Precisión	0.81
Sensibilidad	0.82
Especificidad	0.79
f1-score	0.85
AUC	0.87

Tabla 5.2: Resultados del XGBoost

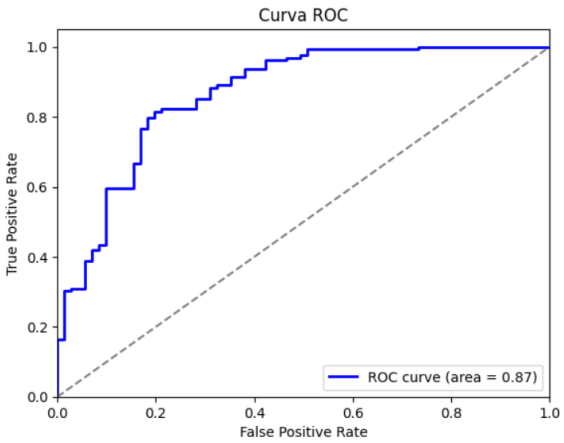

Figura 5.2: Curva ROC del XGBoost

Los métricas de evaluación obtenidas con el modelo XGBoost muestran unos buenos resultados en la clasificación. La precisión alcanzada es de 0.81, que es un poco más baja que en los otros modelos, pero muy cercana. Esto no permite descxartar que con un mejor tunning de los hiperparámetros no se puediese llegar al 0.85 del Random Forest.

Aún así, es un resultado bastante bueno, ya que nuestro principal indicador en este caso es el AUC, que es de momento el mejor AUC de los modelos presentados. Esto se debe a que, aunque la precisión de los otros 2 sea mejor, tienden mucho a una buena sensibilidad pero a una no tan buena especificidad. En el XGBoost, estas dos metricas estan balanceadas, siendo así un modelo un poco menos preciso pero mejor clasificador entre positivos y negativos.

6. Bayesian Trees

El modelo basado en árboles bayesianos ofrece un enfoque probabilístico para la clasificación, incorporando la incertidumbre en sus predicciones. A diferencia de los métodos deterministas como los árboles de decisión tradicionales o XGBoost, los árboles bayesianos generan una distribución posterior sobre las estructuras del árbol y los parámetros, lo que permite una mejor generalización y una mayor robustez frente a la variabilidad de los datos.

Este modelo utiliza técnicas bayesianas para determinar la estructura óptima del árbol, penalizando modelos demasiado complejos y evitando el sobreajuste. Gracias a la estimación de distribuciones de probabilidad en cada nodo, los árboles bayesianos pueden manejar mejor la incertidumbre en los datos y proporcionar predicciones más interpretables.

Además, la naturaleza probabilística del modelo permite la integración de prior knowledge en el proceso de aprendizaje, lo que puede ser ventajoso en escenarios donde la cantidad de datos es limitada o cuando se desea incorporar información previa sobre la distribución de las variables. Estos árboles se han utilizado con éxito en aplicaciones donde la robustez y la interpretación de la incertidumbre son cruciales, como en el modelado de riesgo financiero o la bioinformática.

Es cierto que en la mayoría de documentos divulgativos sobre arboles de decisión, este tipo de árbol no se menciona. Es por eso que la mayor parte de la información aquí presente y del código usado provienen del blog personal y del repositorio de github de Oliver Burkhard.

Estos son algunos de los hiperpar´ametros que podemos editar en este modelo:

- **n_scan**: Número de exploraciones realizadas en cada iteración del algoritmo. Define cuántas soluciones se generan y evalúan antes de seleccionar las mejores. Un valor mayor permite explorar más soluciones pero aumenta el costo computacional.
- **n_keep**: Número de mejores soluciones que se conservan tras cada exploración. Esto ayuda a mantener una buena diversidad de soluciones a lo largo del proceso de optimización, evitando convergencia prematura en óptimos locales.
- **spread_factor**: Factor que controla la dispersión de las soluciones en cada iteración. Valores más altos fomentan la exploración de nuevas áreas del espacio de búsqueda, mientras que valores bajos favorecen la explotación de las soluciones ya encontradas.
- **seed**: Semilla utilizada para la generación de números aleatorios. Esto permite garantizar la reproducibilidad de los resultados, asegurando que se obtengan los mismos valores en distintas ejecuciones.

A continuación, se presentan los hiperparámetros utilizados en el modelo:

Parámetro	Valor
n_scan	10
n_keep	5
spread_factor	0.5
seed	42

Tabla 6.1: Hiperparámetros utilizados en el Bayesian Tree

En este caso, no hay valores por defecto conocidos y se han ido ajustando manualmente para encontrar un buen resultado.

6.1. Evaluación

Seguidamente se presenta la Confusion Matrix para una representación más visual, juntamente con una tabla con las métricas evaluadoras del modelo entrenado y un gráfico de la curva ROC:

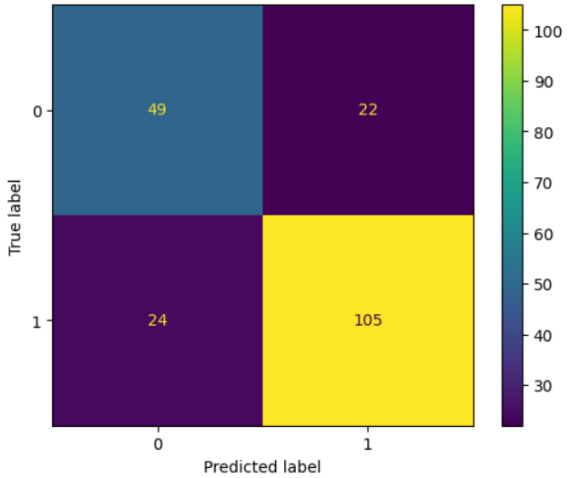

Figura 6.1: Matriz de confusión del Bayesian Tree

Métricas	Resultado
Precisión	0.77
Sensibilidad	0.81
Especificidad	0.69
f1-score	0.82
AUC	0.77

Tabla 6.2: Resultados del Bayesian Tree

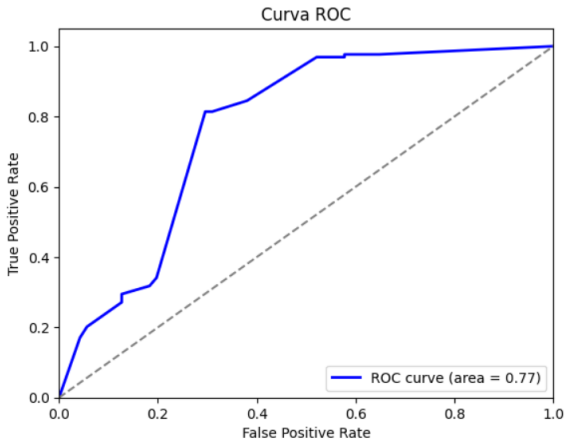

Figura 6.2: Curva ROC del Bayesian tree

COn una sensibilidad mucho mayor que su especificidad, podemos ver como este modelo, también tiende a classificar más resultados positivos de los que realmente hay. Esto es debido, en parte, al mayor número de casos positivos en el dataset. Aún así, no podemos considerar este 0.77 de precisión, ni el 0.77 de AUC como unos malos resultados. Pero también se entiende que para este caso concreto de dataset, no son los mejores resultados y por lo tanto no sería el modelo seleccionado.

7. Conclusiones

Ya se avisaba en el análisis de los datos, como el 62 % de las observaciones eran positivas, indicando una clara mayoría positiva. Esto ha afectado a los modelos, ya que se ha podido ver en todos ellos, como estos clasificaban mejor los positivos que los negativos.

Sabiendo esto, ya se puede discutir acerca de que modelo es mejor. Para esta qüestión no hay una sola respuesta, ya que hay 2 métricas que podemos considerar como evaluadores finales de un modelo, y estas pueden dar vencedores diferentes. Estas métricas son la precisión y el AUC.

La precisión indica el porcentaje de aciertos de un modelo, y el AUC como de buen clasificador es este. En nuestro caso, un modelo que clasifique todos a positivo, habira tenido un 62 % de precisión, pero este tendría un bajo AUC, indicando que es mal classificador.

Métricas	CART	Random Forest	XGB	Bayesian
Precisión	0.85	0.83	0.81	0.77
AUC	0.82	0.90	0.87	0.77

Tabla 7.1: Comparación final de los modelos usados

Para graficar mejor estos resultados compararemos las curvas ROC generadas por estos modelos:

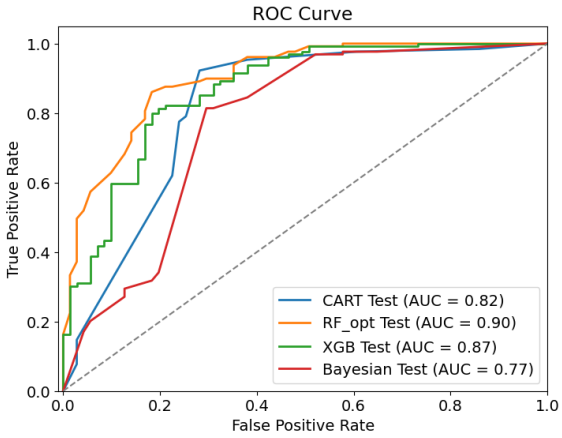

Figura 7.1: Comparación de curvas ROC

Estas son las conclusiones que podemos obtener a partir de los resultados mostrados:

- **Random Forest** presenta el mayor AUC (0,90), lo que indica que es el modelo con mejor capacidad de discriminación entre clases. Sin embargo, su precisión (0,83) es ligeramente inferior a la de CART.
- **CART** tiene la mejor precisión (0,85), pero su AUC (0,82) es más bajo, aunque sigue siendo bueno.
- **XGBoost** muestra un buen balance entre ambas métricas, con un **AUC** de 0,87 y una precisión de 0,81. Su precisión no es tan alta como CART o XGBoost pero su AUC es bastante bueno.
- **El modelo Bayesiano** obtiene el peor rendimiento en ambas métricas (**precisión** = 0,77, **AUC** = 0,77), lo que indica que no es la mejor opción para este contexto.

La conclusión final es que Random Forest destaca como mejor clasificador, seguido por XGBoost, aunque la mejor precisión la tenga CART. Esto significa que para datasets parecidos en proporción de positivos y negativos, CART va a tener muy buenos resultados, pero en caso de datasets diferentes, los mejores clasificadores, como son Random Forest y XGBoost, van a ganar

terreno. En todo caso Random Forest ha destacado en esta comparación de modelos por su buen resultado en ambas métricas.

La mención especial es para el Bayesian tree, ya que no ha tenido mucho éxito en ninguna de las dos métricas, pero esto es debido a que es un modelo muy situacional. Este requiere de un gran dominio sobre el propio modelo y una muy buena perspectiva del contexto, ya que se tiene que saber muy bien en que situaciones usarlo y en cuales no. En esta, por ejemplo, no ha sido un modelo con éxito.

All files used for this project can be found on Github.