

INTELIGENCIA DE NEGOCIO

Práctica 1.

Análisis Predictivo Mediante Clasificación

Alejandro Coman Venceslá



ÍNDICE

| | |
|---|-----------|
| 1. Aprobación de créditos | 3 |
| 1.1. Introducción | 4 |
| 1.2 Procesado de datos | 5 |
| 1.3 Resultados obtenidos | 8 |
| 1.3.1 Random Forest | 9 |
| 1.3.2 Naive Bayes | 10 |
| 1.3.3 Árbol de Decisión | 12 |
| 1.3.4 Gradient Boosted Trees | 13 |
| 1.3.5 Stochastic Gradient Descent | 15 |
| 1.3.6 Curvas ROC | 17 |
| 1.4 Configuración de algoritmos | 18 |
| 1.4.1 Random Forest | 18 |
| 1.4.2 Naive Bayes | 21 |
| 1.4.3 Árbol de Decisión | 23 |
| 1.4.4 Gradient Boosted Trees | 25 |
| 1.4.5 Stochastic Gradient Descent | 27 |
| 1.5 Análisis de resultados | 29 |
| 1.6 Interpretación de los datos | 32 |
| 1.7 Contenido adicional | 33 |
| 1.8 Bibliografía | 33 |
| 2. Segunda Cita | 34 |
| 2.1. Introducción | 34 |
| 2.2. Procesado de datos | 35 |
| 2.3 Resultados obtenidos | 39 |
| 2.3.1 Random Forest | 39 |
| 2.3.2 Naive Bayes | 40 |
| 2.3.3 XGBoost | 42 |
| 2.3.4 Gradient Boosted Trees | 43 |
| 2.3.5 Stochastic Gradient Descent | 45 |
| 2.3.6 Curvas ROC | 47 |
| 2.4 Configuración de algoritmos | 48 |
| 2.4.1 Random Forest | 48 |
| 2.4.2 Naive Bayes | 50 |
| 2.4.3 XGBoost | 50 |
| 2.4.4 Gradient Boosted Trees | 51 |
| 2.4.5 Stochastic Gradient Descent | 53 |
| 2.5 Análisis de resultados | 55 |
| 2.6 Interpretación de los datos | 57 |
| 2.7 Contenido adicional | 58 |
| 2.8 Bibliografía | 58 |
| 3. Tipo de enfermedad eritemato-escamosa | 59 |
| 3.1. Introducción | 59 |
| 3.2. Procesado de datos | 60 |

Alejandro Coman Venceslá

| | |
|---------------------------------|----|
| 2.3 Resultados obtenidos | 63 |
| 3.3.1 KNN | 63 |
| 3.3.2 Gradient Boosted Trees | 64 |
| 3.3.3 Random Forest | 65 |
| 3.3.4 Árbol Decisión | 66 |
| 3.3.5 Support Vector Machine | 67 |
| 3.4 Configuración de algoritmos | 68 |
| 3.4.1 KNN | 68 |
| 3.4.2 Gradient Boosted Trees | 69 |
| 3.4.3 Random Forest | 69 |
| 3.4.4 Árbol Decisión | 69 |
| 3.4.5 Support Vector Machine | 70 |
| 3.5 Análisis de resultados | 70 |
| 3.6 Interpretación de los datos | 71 |
| 3.7 Contenido adicional | 73 |
| 3.8 Bibliografía | 73 |

1. Aprobación de créditos

1.1. Introducción

Este dataset recopila información sobre la aprobación de préstamos en función de varias características individuales de los solicitantes. Su objetivo es predecir si una solicitud de préstamo será aprobada.

Consideraciones generales:

- **Dimensiones:** Con un total de 12 atributos distintos, tenemos 32,581 instancias, lo cual lo hace un conjunto de datos grande y adecuado para el análisis detallado.
- **Tipos de variables:** Incluye tanto variables numéricas como categóricas. 8 son numéricas y 4 categóricas.
- **Desbalanceo de clases:** Podemos ver un claro ejemplo de desbalanceo de clases ya que "loan_status" tiene casi un 80% (25,473 instancias) con valor 0 (rechazado). Será importante tener esto en cuenta posteriormente para entrenar los algoritmos.
- **Valores perdidos:** Las variables "person_emp_length" y "loan_int_rate" tienen algunas instancias con valor nulo (895 y 3,116 respectivamente). Es importante tener en cuenta esto para el posterior preprocesamiento de los datos para los algoritmos, en caso de que requieran que todas las filas no tengan campos vacíos.

Otras consideraciones acerca de los valores de los datos:

- **Edad e ingresos:** la mayoría de solicitudes provienen de personas de entre 20 y 30 años, con ingresos entre 9,600\$ y 65,500\$
- **Intención del solicitante del préstamo:** la mayoría de solicitantes son inquilinos con necesidades financieras a corto plazo, pues pretenden emplear el préstamo para gastos personales y médicos.
- **Historial crediticio:** la mayoría de los solicitantes tienen un historial crediticio corto y una gran parte de estos han presentado un impago.
- **Grados del préstamo y tipos de interés:** casi todos los préstamos tienen grado "C" y "D", alineándose con tipos de interés altos (11.14% - 16.02%).
- **Cantidad del préstamo vs. salario:** la mayoría de los aplicantes piden más dinero que su salario.

Para realizar la predicción de este problema he seleccionado los siguientes algoritmos:

1. **Random Forest:** Al combinar múltiples árboles de decisión, son robustos y presentan un buen rendimiento al trabajar con conjuntos de datos mixtos como el de este problema.
2. **Naive Bayes:** Este algoritmo es bastante rápido y eficiente, lo cual viene bien, contando con que tenemos una gran cantidad de instancias. Además, también maneja bien los conjuntos de datos mixtos.

Alejandro Coman Venceslá

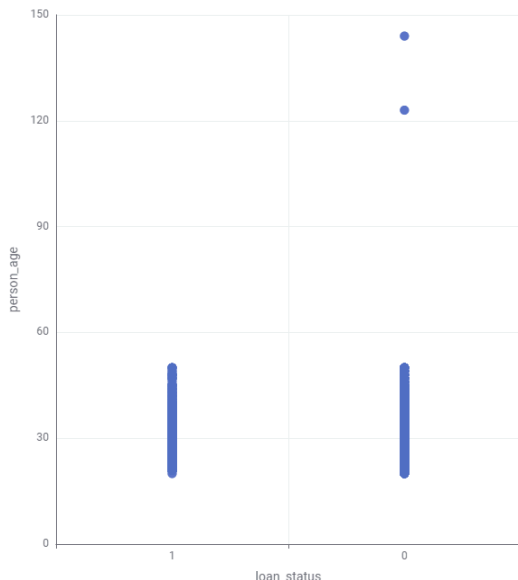
3. **Árbol de Decisión:** Utiliza una estructura de árbol para representar decisiones y sus posibles consecuencias. Es fácil de interpretar y visualizar, lo que ayuda a comprender cómo diferentes variables afectan la aprobación de créditos.
4. **Gradient Boosted Trees:** Método de ensamble que construye secuencialmente árboles de decisión, donde cada nuevo árbol intenta corregir los errores del anterior. Es conocido por su alto rendimiento predictivo y su capacidad para capturar relaciones no lineales y patrones complejos en los datos. En el dataset de aprobación de créditos, puede manejar eficientemente variables mixtas y el desbalance de clases.
5. **SGD:** SGD es un algoritmo de optimización ampliamente utilizado en el aprendizaje automático y la clasificación. Presenta un buen rendimiento en problemas de clasificación binaria.

1.2 Procesado de datos

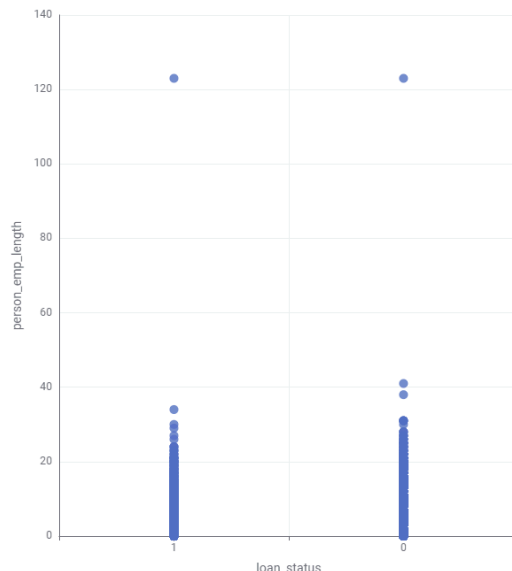
A modo de preprocesamiento general para los algoritmos, he decidido considerar una serie de decisiones:

En primer lugar he procedido a visualizar los datos con el nodo **Scatter Plot**. Poniendo la clase objetivo “loan status” en el eje horizontal y luego visualizando el resto de variables en el eje vertical, una a una. He podido comprobar que en algunas variables podemos encontrar valores que tienen poco sentido.

Scatter Plot



Scatter Plot



Figuras 1 y 2. Scatter plots de “person_age” y “person_emp_length” frente a “loan_status”.

En las Figuras 1 y 2 estamos mostrando las variables “person_age” y “person_emp_length”, frente a la variable objeto de clasificación. Podemos comprobar que estas dos variables presentan valores superiores a 120 años, lo cual no tiene sentido. Entonces aquí podemos optar por dejar estas instancias o borrarlas. Además de acotar estos dos atributos, también

Alejandro Coman Venceslá

podemos acotar otros atributos como el ingreso de cada persona, o incluso acotar más todavía el de la edad. Recordemos que la mayoría de aplicantes tienen entre 20 y 30 años y un ingreso entre 9,600 y 65,500\$.

Posteriormente, para tratar los valores faltantes en algunas instancias en las clases “person_emp_length” y “loan_int_rate”, mediante el nodo **Missing Value**, he considerado dos opciones: Rellenar los valores mediante el valor más frecuente para las variables categóricas y realizar una media en valores numéricas o directamente eliminar las instancias que tengan valores vacíos.

De igual manera que podemos eliminar las filas con aquellos valores vacíos en estos dos atributos, también podemos eliminar directamente estas dos columnas del modelo.

Por último, también podemos considerar normalizar todos los datos antes de procesarlos por los algoritmos.

Tras evaluar las distintas alternativas, no he podido encontrar casi ninguna diferencia en el rendimiento de los distintos algoritmos. Es por ello que, por temas de coste computacional, he decidido aplicar el preprocesamiento que me deje con el menor número de instancias en el modelo.

Borramos las instancias con valores extremos en la edad, historial crediticio y salario de cada persona, mediante el nodo **Row Filter** y luego optamos por eliminar los atributos mencionados anteriormente con el nodo **Column Filter**. Row Filter queda de esta manera:

La imagen muestra la configuración de cinco criterios de filtrado en el nodo Row Filter, organizados en dos columnas. Cada criterio tiene un título (Criterion 1 a 5), un botón para moverlo (flechas arriba y abajo) y un botón para borrarlo (icono de basura).

- Criterion 1:** Filter column: person_age, Operator: Less than or equal, Value: 35.
- Criterion 2:** Filter column: person_age, Operator: Greater than or equal, Value: 20.
- Criterion 3:** Filter column: person_emp_length, Operator: Less than or equal, Value: 90.
- Criterion 4:** Filter column: person_income, Operator: Greater than or equal, Value: 8000.
- Criterion 5:** Filter column: person_income, Operator: Less than or equal, Value: 70000.

Figuras 3 y 4. Nodo Row Filter para acotar los valores de algunos atributos del modelo.

En las Figuras 3 y 4 estoy aplicando filtros a los registros ya que estoy tratando de eliminar primero el ruido del conjunto de datos. Cabe destacar que no he acotado los valores exactamente por los indicados en la descripción del conjunto de datos, sino que he dejado un rango de valores ligeramente más amplio.

Alejandro Coman Venceslá

Con estos cambios, he pasado de 12 atributos con 32,581 instancias a 10 atributos con 19,140 instancias, habiendo reducido el número de valores del modelo de 390,972 a 191,400 (aproximadamente la mitad) consiguiendo un rendimiento prácticamente sin ninguna diferencia de si hubiéramos conservado el conjunto de datos entero. Esto será un ahorro computacional bastante grande.

El flujo de datos en el preprocesamiento queda de la siguiente manera:

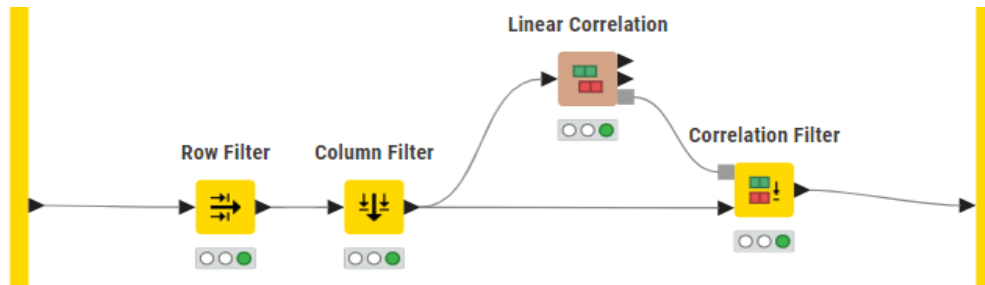


Figura 5. Nodo de preprocesamiento general para la predicción de aprobación de créditos.

El resto del flujo de datos se estructura de la siguiente manera:

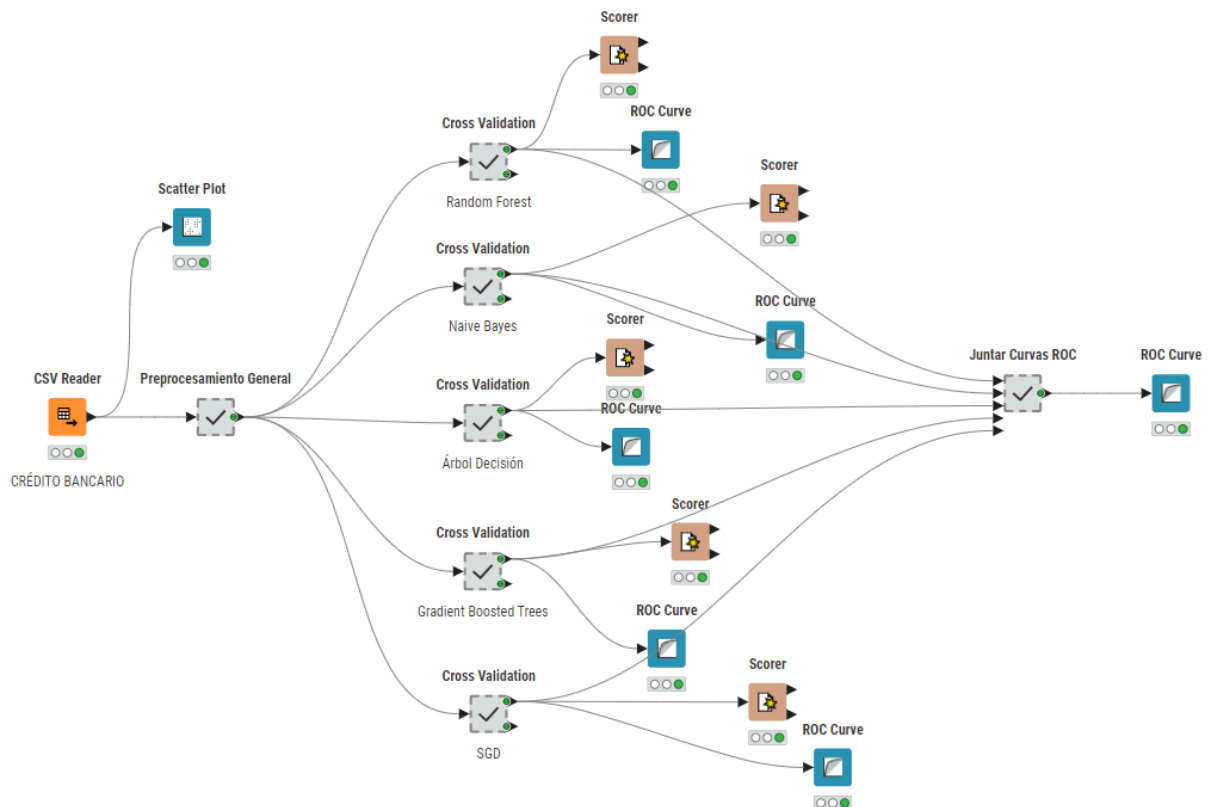


Figura 6. Vista general del flujo del trabajo para el problema de la aprobación de créditos.

Tras leer los datos del archivo CSV y aplicar el preprocesamiento anteriormente mencionado, mandaremos estos datos a cada algoritmo, el cual tendrá una validación cruzada con 5 validaciones. Todos los nodos de validación cruzada (X-Partitioner y X-Aggregator) tienen la siguiente configuración:

Alejandro Coman Venceslá

Number of validations: 5

Linear sampling: ☐

Random sampling: ☐

Stratified sampling: ☒

Class column: loan_status

☒ Random seed: 123456

Leave-one-out: ☐

Target column: loan_status

Prediction column: Prediction (loan_status)

☒ Add column with fold id

Figuras 7 y 8. Configuración de los nodos X-Partitioner (izquierda) y X-Aggregator (derecha) para la validación cruzada.

En las Figuras 7 y 8 muestro la configuración de los nodos encargados de la validación cruzada. Se ha utilizado “stratified sampling” basado en la clase objetivo (loan_status en este caso), para intentar aminorar el problema del desbalanceo de las clases.

Una vez terminan de ejecutarse todos los algoritmos, para poder ver los resultados, se usa para cada algoritmo los nodos **Scorer** y **ROC Curve**. En adición se ha construido otro metanodo para unir todas las columnas de las predicciones en una sola tabla y así poder ver todas las curvas ROC en un solo nodo. Esto lo podemos apreciar en la Figura 9.

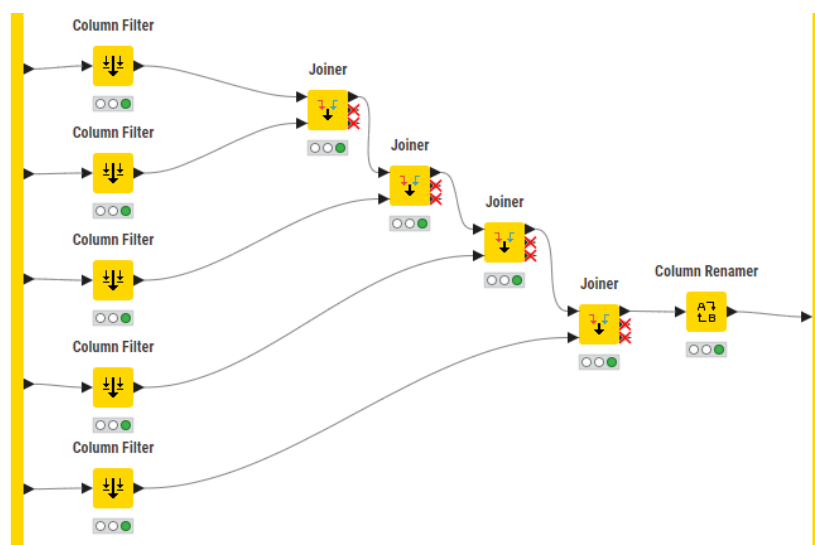


Figura 9. Metanodo para juntar las columnas de las predicciones de los distintos algoritmos y representarlas todas las curvas ROC juntas.

1.3 Resultados obtenidos

A continuación vamos a comprobar los resultados que hemos obtenido para cada uno de los algoritmos, habiendo aplicado este preprocesamiento general y habiendo dejado la configuración de los algoritmos por defecto. Mostraremos las matrices de confusión y las tablas de estadísticas de los nodos Scorer, además de las curvas ROC individuales. También podremos analizar las primeras métricas que obtenemos de cada algoritmo.

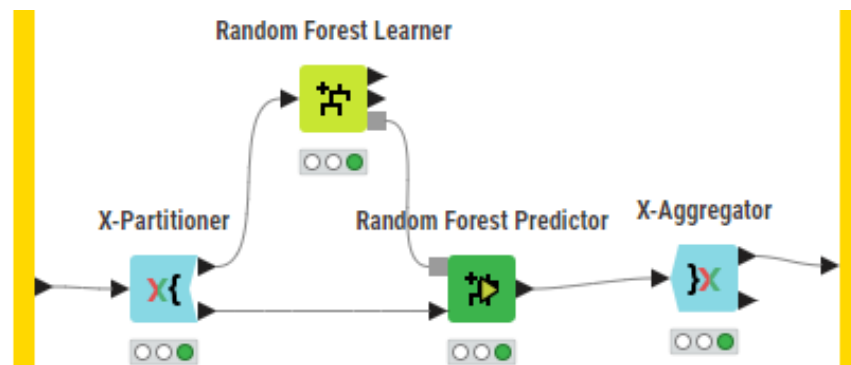
1.3.1 Random Forest

Figura 10. Metanodo de la validación cruzada para el algoritmo "Random Forest".

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 3795 | 1321 |
| <i>loan_status</i> = 0 | 305 | 13719 |

Tabla 1. Matriz de confusión de Random Forest para la aprobación de créditos.

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|-------|------|-------|------|--------|-----------|-------------|-----------|-------|-------|
| 1 | 3795 | 305 | 13719 | 1321 | 0.741 | 0.925 | 0.741 | 0.978 | 0.823 | |
| 0 | 13719 | 1321 | 3795 | 305 | 0.978 | 0.912 | 0.978 | 0.741 | 0.944 | |
| Overall | | | | | | | | | | 0.915 |

Tabla 2. Estadísticas de precisión de Random Forest para la aprobación de créditos.

A primera vista en las Tablas 1 y 2, podemos ver como el desbalanceo de clases, siendo la clase negativa la mayoritaria, tiene su efecto a la hora de predecir. Esto podemos comprobarlo con la cantidad de falsos negativos que tenemos, que es claramente significativamente superior a los falsos positivos. A pesar de que tenemos un Accuracy de 0.915, lo cual es alto, esta medida de por sí sola no es suficiente para poder afirmar que el clasificador es bueno. La debemos complementar con más medidas de rendimiento.

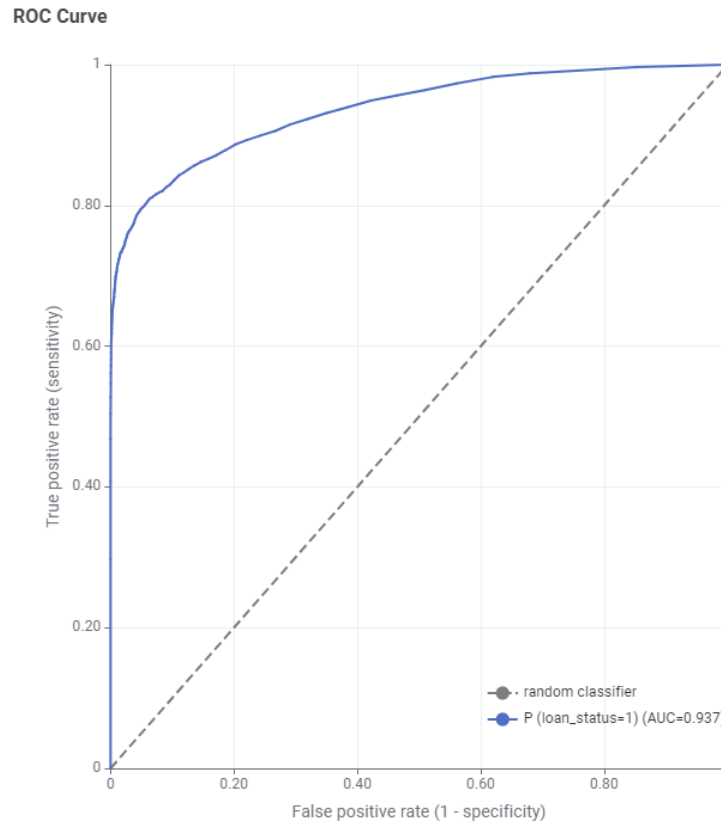


Figura 11. Curva ROC de Random Forest para la aprobación de créditos.

En la Figura 11 mostramos la curva ROC de las predicciones de este algoritmo. Podemos ver que tiene una forma bastante buena. Además el AUC es de 0.937, lo cual indica un rendimiento muy bueno. Podemos afirmar que este clasificador es bastante bueno de por sí solo, aún con el problema del desbalanceo de clases. No obstante, se intentarán hacer algunos ajustes más adelante para comprobar si se puede mejorar esta cifra.

1.3.2 Naive Bayes

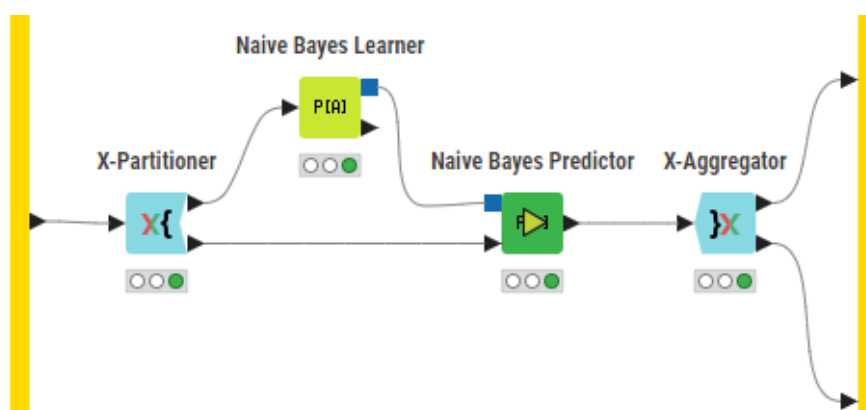


Figura 12. Metanodo de la validación cruzada para el algoritmo "Naive Bayes".

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 3025 | 2091 |
| <i>loan_status</i> = 0 | 946 | 13078 |

Tabla 3. Matriz de confusión de Naive Bayes para la aprobación de créditos.

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|-------|------|-------|------|--------|-----------|-------------|-----------|-------|-------|
| 1 | 3025 | 946 | 13078 | 2091 | 0.591 | 0.761 | 0.591 | 0.932 | 0.666 | |
| 0 | 13078 | 2091 | 3025 | 946 | 0.932 | 0.862 | 0.933 | 0.591 | 0.896 | |
| Overall | | | | | | | | | | 0.841 |

Tabla 4. Estadísticas de precisión de Naive Bayes para la aprobación de créditos.

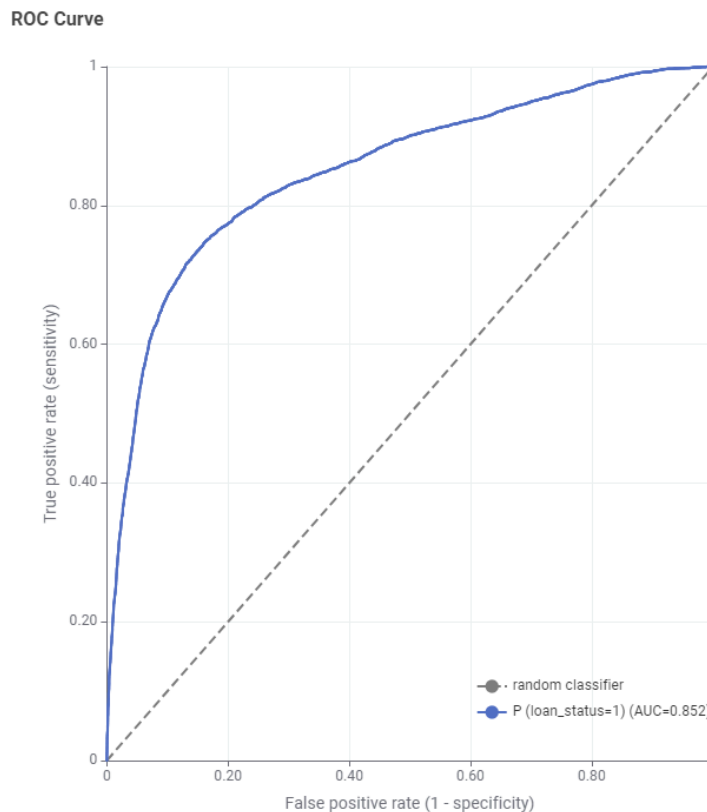


Figura 13. Curva ROC de Naive Bayes para la aprobación de créditos.

En este escenario, tras ver las Tablas 3 y 4 y la Figura 13, podemos observar como el algoritmo Naive Bayes tiene también un Accuracy alto (0.841) y un AUC alta también, como se puede apreciar en la curva ROC (0.852). No obstante estas medidas son algo peores que con Random Forest. Quizá esto se pueda deber al hecho de que Naive Bayes asume independencia entre todas las variables del modelo. Esto provoca que las probabilidades de cada clase estén, a priori, sesgadas hacia la clase mayoritaria. Veremos más adelante si podemos aminorar estas ineficiencias.

1.3.3 Árbol de Decisión

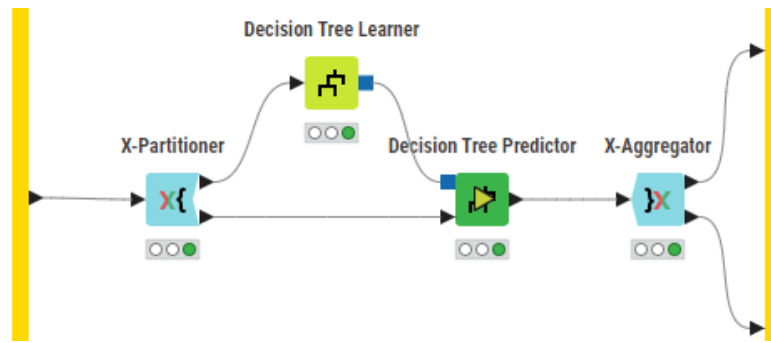


Figura 14. Metanodo de la validación cruzada para el algoritmo "Decisión Tree".

Figura 15. Configuración por defecto del nodo "Decision Tree Learner"

En la Figura 15 podemos ver que, por defecto, estamos utilizando el Índice de Gini para buscar los atributos más fuertes y que no estamos realizando ninguna poda en el Árbol.

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 3863 | 1250 |
| <i>loan_status</i> = 0 | 1114 | 12904 |

Tabla 5. Matriz de confusión del Árbol de Decisión para la aprobación de créditos.

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|-------|------|-------|------|--------|-----------|-------------|-----------|-------|-------|
| 1 | 3863 | 1114 | 12904 | 1250 | 0.756 | 0.776 | 0.756 | 0.921 | 0.766 | |
| 0 | 12904 | 1250 | 3863 | 1114 | 0.921 | 0.912 | 0.921 | 0.756 | 0.916 | |
| Overall | | | | | | | | | | 0.876 |

Tabla 6. Estadísticas de precisión de Árbol de Decisión para la aprobación de créditos.

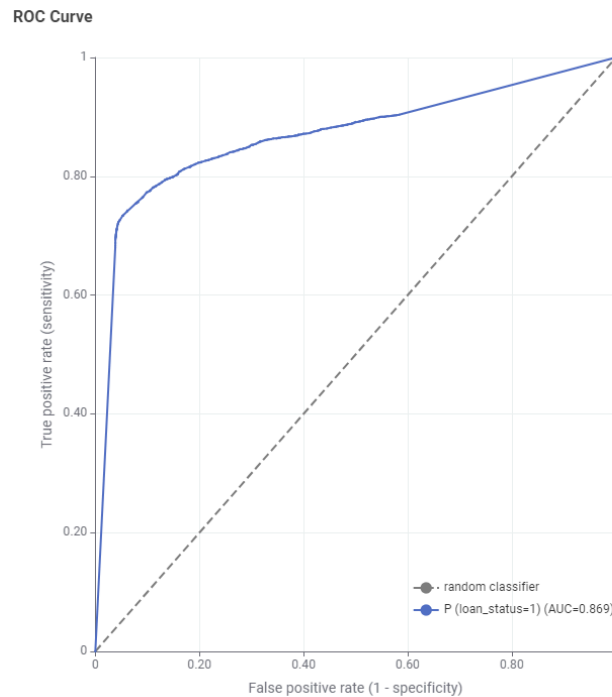


Figura 16. Curva ROC del Árbol de Decisión para la aprobación de créditos.

En este escenario, tras observar las Tablas 5 y 6 y las Figuras 15 y 16, podemos ver cómo el algoritmo del Árbol de Decisión tiene también un Accuracy alto (0.876) y un AUC alta también, como se puede apreciar en la curva ROC (0.869). Son métricas que indican un buen rendimiento, pero aún mejorable. Son también peores que Random Forest, ya que, un árbol de decisión normal tiende a sobre ajustarse más al no contar con la capacidad de Random Forest de promediar resultados de varios árboles. Además no estamos realizando ninguna poda en el árbol y tendremos que ver si el Índice de Gini nos proporciona el modelo más óptimo.

Es por ello que, intentaremos igualmente ajustar el algoritmo más adelante para comprobar si podemos mejorar las cifras.

1.3.4 Gradient Boosted Trees

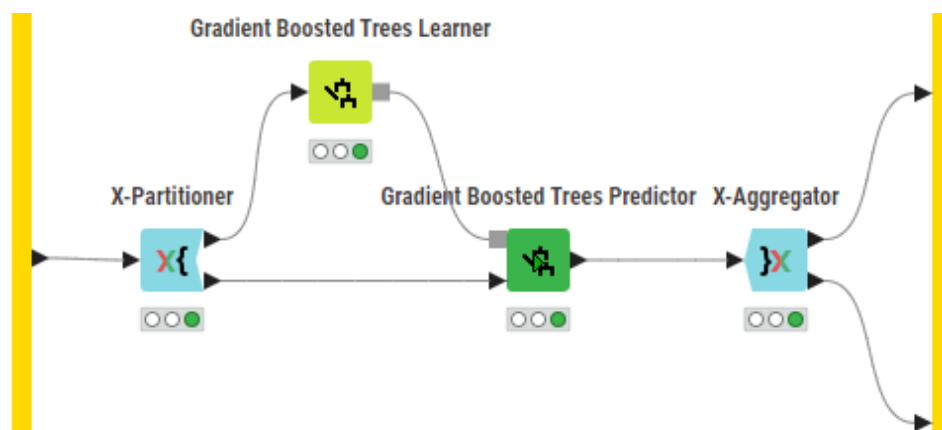


Figura 17. Metanodo de la validación cruzada para el algoritmo "Gradient Boosted Trees".

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 3794 | 1322 |
| <i>loan_status</i> = 0 | 231 | 13793 |

Tabla 7. Matriz de confusión de Gradient Boosted Trees para la aprobación de créditos.

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|-------|------|-------|------|--------|-----------|-------------|-----------|-------|-------|
| 1 | 3794 | 231 | 13793 | 1322 | 0.742 | 0.943 | 0.742 | 0.984 | 0.830 | |
| 0 | 13793 | 1322 | 3794 | 231 | 0.984 | 0.913 | 0.984 | 0.742 | 0.947 | |
| Overall | | | | | | | | | | 0.919 |

Tabla 8. Estadísticas de precisión de Gradient Boosted Trees para aprobación de créditos.

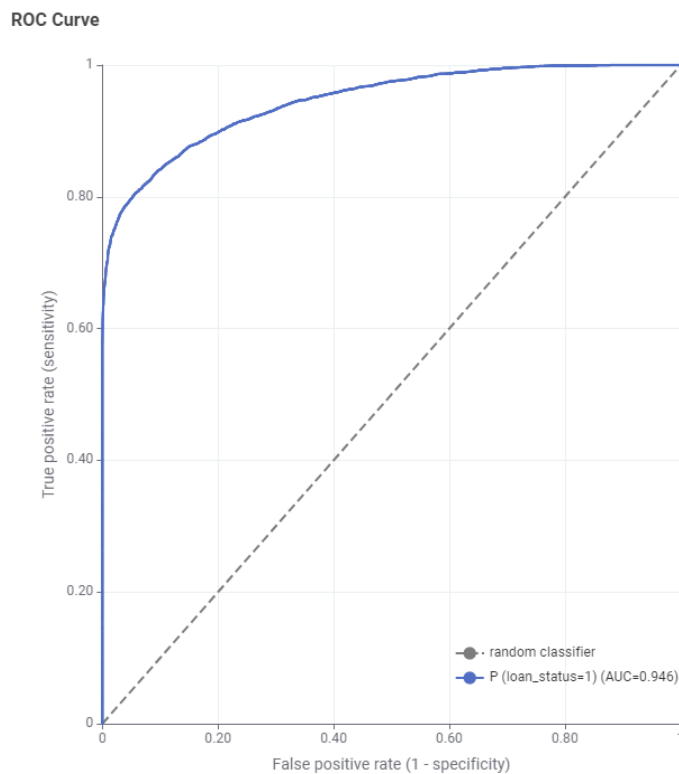


Figura 18. Curva ROC de Gradient Boosted Trees para la aprobación de créditos.

Como podemos observar en las Tablas 7 y 8 y en la Figura 18, hemos obtenido en este escenario una precisión con valor 0.919 y un área debajo de la curva (AUC) con valor 0.946, las cuales son muy buenas métricas y las mejores que hemos obtenido hasta ahora. Podemos afirmar que el algoritmo Gradient Boosted Trees es un buen clasificador para este problema ya que es un método de Boosting, en el cual se generan varios árboles y cada árbol intenta aprender del anterior.

Alejandro Coman Venceslá

Intentaremos igualmente mejorar el rendimiento más adelante ya que seguimos viendo indicios de un sesgo en el modelo debido al desbalanceo de clases, representado en una alta cantidad de falsos negativos.

1.3.5 Stochastic Gradient Descent

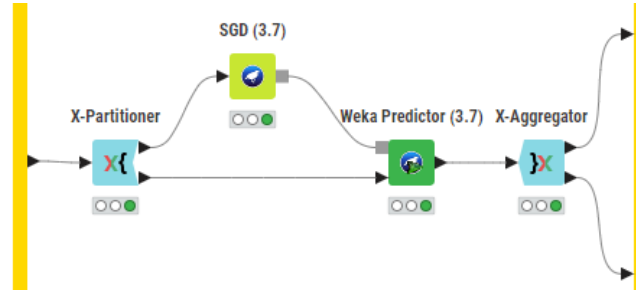


Figura 19. Metanodo de la validación cruzada para el algoritmo “Stochastic Gradient Descent”

Figura 20. Configuración por defecto del nodo “SGD”

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 3060 | 2056 |
| <i>loan_status</i> = 0 | 840 | 13184 |

Tabla 9. Matriz de confusión de SGD para la aprobación de créditos.

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|-------|------|-------|------|--------|-----------|-------------|-----------|-------|-------|
| 1 | 3060 | 840 | 13184 | 2056 | 0.598 | 0.785 | 0.598 | 0.940 | 0.679 | |
| 0 | 13184 | 2056 | 3060 | 840 | 0.940 | 0.865 | 0.940 | 0.598 | 0.901 | |
| Overall | | | | | | | | | | 0.849 |

Tabla 10. Estadísticas de precisión de SGD para aprobación de créditos.

Fijándonos solamente en las Tablas 9 y 10 podemos ver que este algoritmo nos presenta un peor rendimiento que el resto. No solamente porque SGD sea un algoritmo que tiende bastante a sobreajustarse fácilmente, sino que tampoco trata de manera óptima el desbalanceo de clases. Aunque el Accuracy sea bastante bueno, ya que es de 0.849, no nos sirve de por sí para poder llegar a una conclusión definitiva.

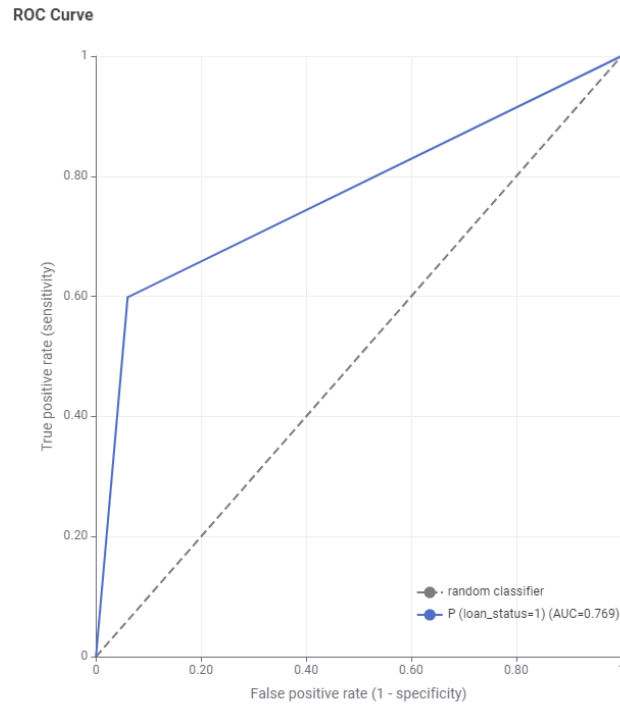


Figura 21. Curva ROC de SGD para la aprobación de créditos.

Mostrando también en la Figura 21 la curva ROC para el algoritmo SGD, y obteniendo un valor AUC de 0.769, vemos que, si bien no es muy malo del todo, se queda bastante lejos de ser un clasificador óptimo. Al tratarse SGD de una red neuronal, una de las desventajas que éstas nos presentan es que sus modelos son muy poco interpretables. Es por ello que más adelante tendremos que intentar mejorar este clasificador mediante el ajuste por prueba y error de varios parámetros del nodo.

Alejandro Coman Venceslá

1.3.6 Curvas ROC

A continuación mostramos todas las curvas ROC juntas.

ROC Curve

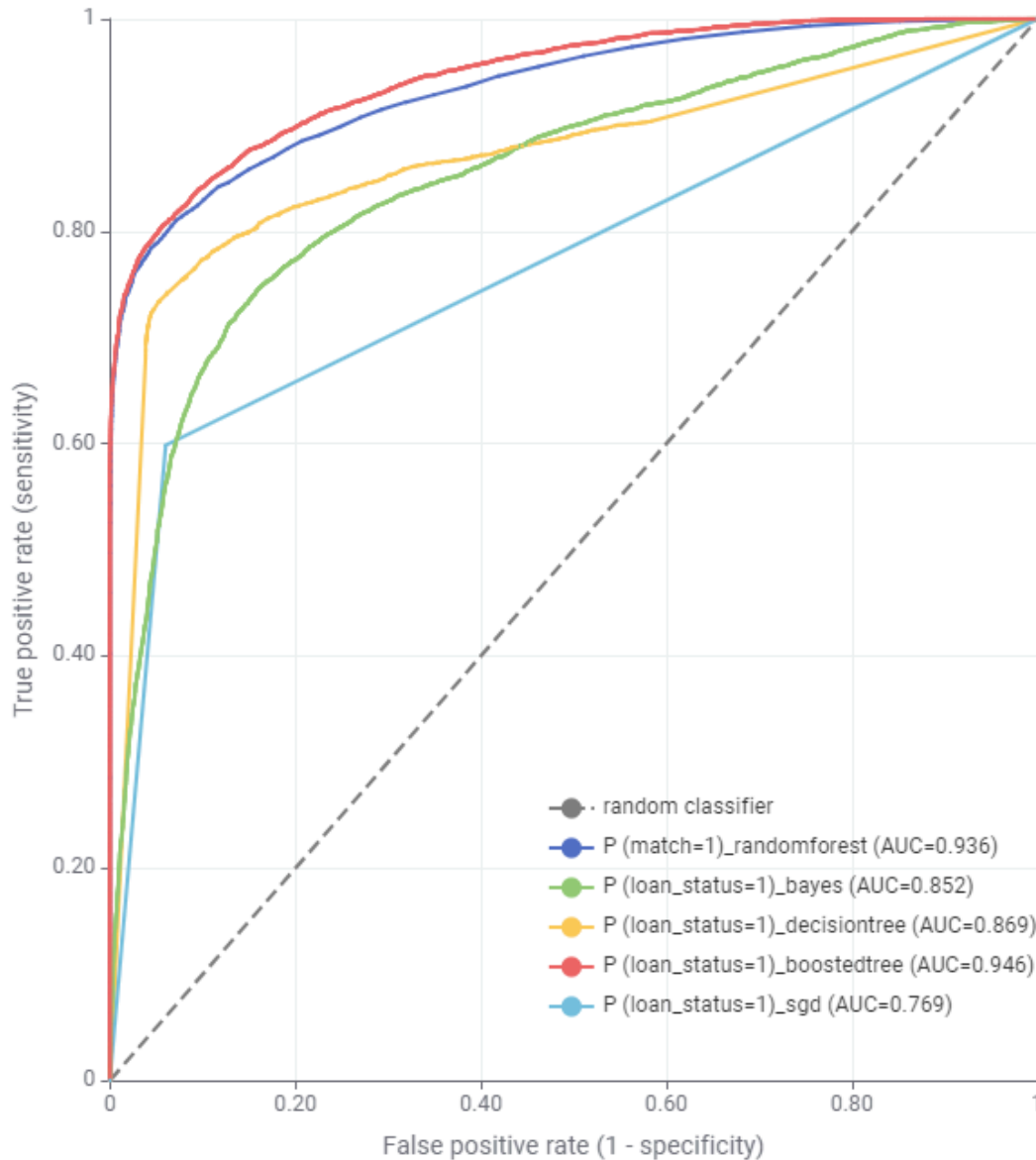


Figura 22. Curvas ROC de los 5 algoritmos usados para el problema de la aprobación de créditos.

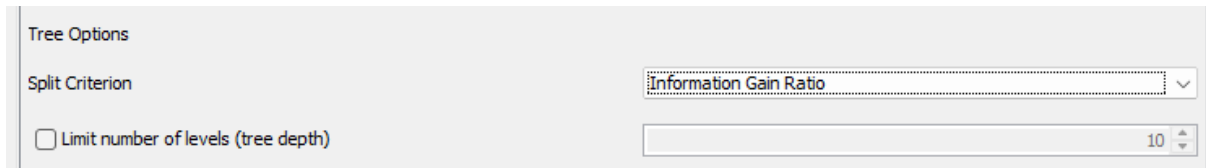
En la Figura 22 mostramos las curvas ROC de los 5 algoritmos que se han empleado para este primer conjunto de datos. En general, hemos propuesto unos algoritmos que dan bastante buenos resultados, a excepción de SGD. Aquellos que más destacan son Random Forest y Gradient Boosted Tree.

Esto se puede deber a que son métodos de ensamble, los cuales, en general, proporcionan mejores resultados que otros que no lo son.

1.4 Configuración de algoritmos

1.4.1 Random Forest

A pesar de haber tenido este algoritmo muy buen rendimiento, vamos a cambiar algunos parámetros para ver si conseguimos mejorar los resultados de este clasificador. Vamos a tocar específicamente “Split Criterion” y “Tree Depth”, que se encargan de decidir cómo se va dividiendo el árbol y cuál será la profundidad máxima que podrá tener éste.



Tree Options

Split Criterion: Information Gain Ratio

☐ Limit number of levels (tree depth) 10

Figura 23. Parámetros a configurar de Random Forest

Como mostramos en la Figura 23, vamos a evaluar las tres alternativas para Split Criterion (Information Gain, Information Gain Ratio y Gini Index) y los valores de 5, 10 y 15 para la profundidad del árbol y ver cómo cambian las métricas de “Accuracy” y “AUC”. Nuestros valores por defecto son “Information Gain Ratio” y “10”.

| PRECISIÓN | | | | AUC | | | |
|------------------------------|-------|--------------|-------|------------------------------|-------|--------------|-------|
| Split Criterion / tree depth | 5 | 10 | 15 | Split Criterion / tree depth | 5 | 10 | 15 |
| Information Gain | 0,905 | 0,916 | 0,916 | Information Gain | 0,888 | 0,935 | 0,928 |
| Information Gain Ratio | 0,903 | 0,918 | 0,917 | Information Gain Ratio | 0,899 | 0,937 | 0,922 |
| Gini Index | 0,905 | 0,916 | 0,917 | Gini Index | 0,899 | 0,932 | 0,927 |

Tabla 11. Resultados al modificar parámetros en Random Forest.

Viendo los resultados comparativos en la Tabla 11, podemos observar que si reducimos la profundidad máxima con la que el árbol se puede expandir, éste no aprenderá bien de todo los datos, por lo que no podrá generalizar tan bien. No obstante, si aumentamos este parámetro tampoco percibimos una mejora en el rendimiento, sino una pequeña disminución. Esto se puede deber a que estamos ante un sobreajuste ya que tendríamos un modelo demasiado fuerte.

Si contamos con una profundidad de 10, el criterio de división “Information Gain Ratio” es el mejor de los tres. No obstante, esta es la configuración por defecto que traía el nodo.

En adición vamos a intentar resolver el problema que teníamos anteriormente con el desbalanceo de clases. Contamos con dos opciones, que es la de llevar a cabo un sobre muestreo de la clase minoritaria con el nodo SMOTE o un submuestreo de la clase mayoritaria.

Podríamos aplicar el nodo SMOTE, no obstante esto debemos hacerlo únicamente al conjunto de entrenamiento dentro de cada fold en la validación cruzada. Ejecutar el nodo SMOTE es bastante costoso y lento, mucho más si tenemos que hacerlo repetidas veces.

Alejandro Coman Venceslá

Es por ello que será un submuestreo de la clase mayoritaria lo que aplicaremos. Esto lo realizaremos como se ha procedido en la siguiente figura:

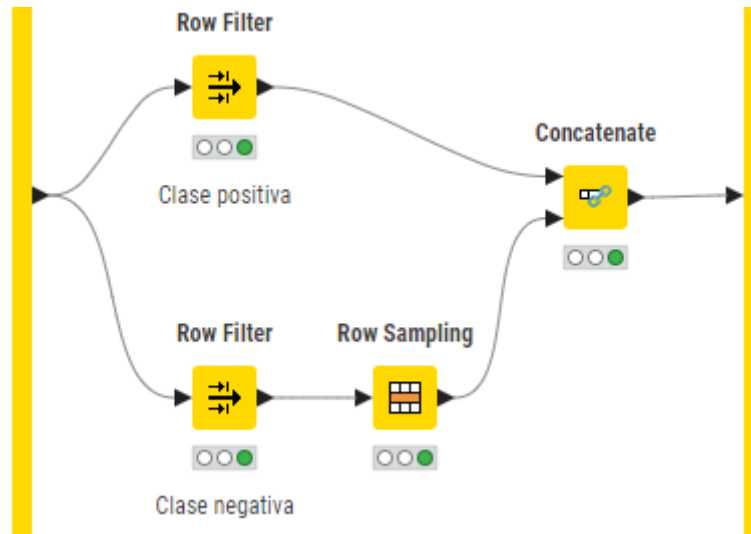


Figura 24. Flujo de datos para el submuestreo de la clase mayoritaria.

Como mostramos en la Figura 24, hemos dividido los datos en dos tablas, cada una conteniendo los registros con clase positiva y negativa, respectivamente. Hemos comprobado cuántos registros había en la clase minoritaria (unos 5000) y hemos filtrado registros en la clase mayoritaria (negativa) tal que ahora las dos clases tendrán el mismo número de registros. Finalmente, hemos unido las dos tablas de nuevo.

Realizando estos cambios podemos ahora hacer una comparativa del rendimiento realizando el submuestreo y sin realizarlo.

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,918 | 0.917 |
| AUC | 0,937 | 0.964 |

Tabla 12. Comparación del rendimiento del algoritmo Random Forest, entre aplicar o no submuestreo a la clase mayoritaria.

Si bien la precisión no ha aumentado, podemos concluir que al aplicar el submuestreo de la clase mayoritaria, el valor del AUC aumenta significativamente. Este enfoque será el que vayamos a aplicar en el resto de algoritmos, ya que en el apartado 1.3 de esta memoria, hemos concluido que todos los algoritmos tienen problemas al tratar con datos con clases desbalanceadas.

Mostramos los nuevos resultados tras aplicar estas optimizaciones.

Alejandro Coman Venceslá

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 4478 | 638 |
| <i>loan_status</i> = 0 | 213 | 4903 |

Tabla 13. Matriz de confusión de Random Forest optimizado.

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 4478 | 213 | 4903 | 638 | 0.875 | 0.955 | 0.875 | 0.958 | 0.913 | |
| 0 | 4903 | 638 | 4478 | 213 | 0.958 | 0.885 | 0.958 | 0.875 | 0.920 | |
| Overall | | | | | | | | | | 0.917 |

Tabla 14. Estadísticas de precisión de Random Forest optimizado.

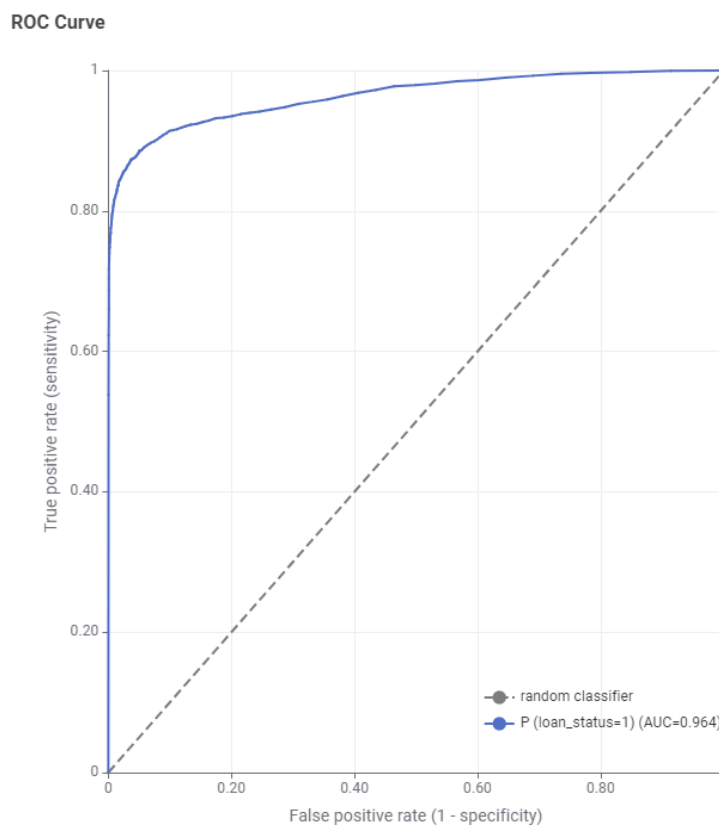


Figura 25. Nueva curva ROC para Random Forest optimizado.

1.4.2 Naive Bayes

Tras haber experimentado con modificaciones en todos los parámetros que este nodo presenta y ver que no hay prácticamente ningún cambio he optado por realizar cambios distintos a tocar los parámetros del algoritmo. Vamos a comprobar si el rendimiento mejora al introducir en el flujo de datos los nodos de **Normalization** y **One To Many**.

Lo que hace One To Many es procesar las variables categóricas tal que por cada valor posible que tenga esa variable categórica, crea una columna nueva en la tabla. Si una instancia tenía un valor dado de esa variable categórica, la fila nueva creada de ese valor se pondrá a 1 y el resto de filas a 0. (Hemos convertido 1 variable de N clases a N variables binarias).

Es imprescindible excluir la clase objetivo del modelo. Ya que si la incluimos, la va a dividir también y entrenar el modelo sobre esta variable. Estaríamos ante un sobreajuste (se puede comprobar que si no la excluimos vamos a tener siempre una precisión y AUC de 1).

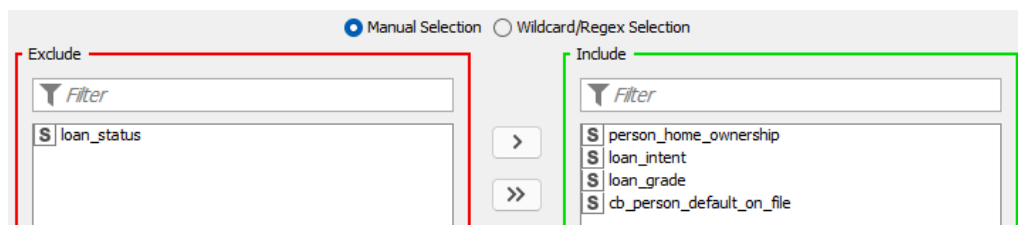


Figura 26. Configuración del nodo One To Many, excluyendo la clase objetivo.

Nuestra configuración por defecto es no aplicar ni Normalización ni One To Many.

| PRECISIÓN | | | AUC | | |
|-----------------------------|-------|---------------------|-----------------------------|-------|---------------------|
| Normalización / One To Many | Si | No | Normalización / One To Many | Si | No |
| Sí | 0,8 | <u>0,839</u> | Si | 0,817 | <u>0,859</u> |
| No | 0,794 | 0,852 | No | 0,8 | 0,852 |

Tabla 15. Resultados al modificar parámetros en Naive Bayes

Vemos que, aunque la precisión sea ligeramente mejor dejándolo por defecto, hay una mejora en el AUC para cuando aplicamos normalización, lo cual tiene sentido ya que normalizar los datos, generalmente, suele estabilizar el entrenamiento de los algoritmos, llegando a una convergencia de manera más rápida y proporcionando mejores resultados. Es por ello que vamos a optar por dejar la normalización en este algoritmo. En adición, vamos a aplicar también el submuestreo de la clase mayoritaria en este algoritmo y ver si mejora el rendimiento del clasificador:

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,839 | 0.812 |
| AUC | 0,859 | 0.904 |

Tabla 16. Comparación del rendimiento del algoritmo Naive Bayes, entre aplicar o no submuestreo a la clase mayoritaria

Sabiendo cual es la mejor configuración de las que hemos estudiado vamos a mostrar de nuevo los resultados:

| Clase / Predicción | loan_status (predicción) = 1 | loan_status (predicción) = 0 |
|--------------------|------------------------------|------------------------------|
| loan_status = 1 | 3655 | 1576 |
| loan_status = 0 | 343 | 4773 |

Tabla 17. Matriz de confusión de Naive Bayes optimizado.

| loan_status | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|-------------|------|------|------|------|--------|-----------|-------------|-----------|-------|-------|
| 1 | 3545 | 336 | 4780 | 1571 | 0.693 | 0.913 | 0.693 | 0.934 | 0.788 | |
| 0 | 4780 | 1571 | 3545 | 336 | 0.9348 | 0.753 | 0.934 | 0.693 | 0.834 | |
| Overall | | | | | | | | | | 0.814 |

Tabla 18. Estadísticas de precisión de Naive Bayes optimizado.

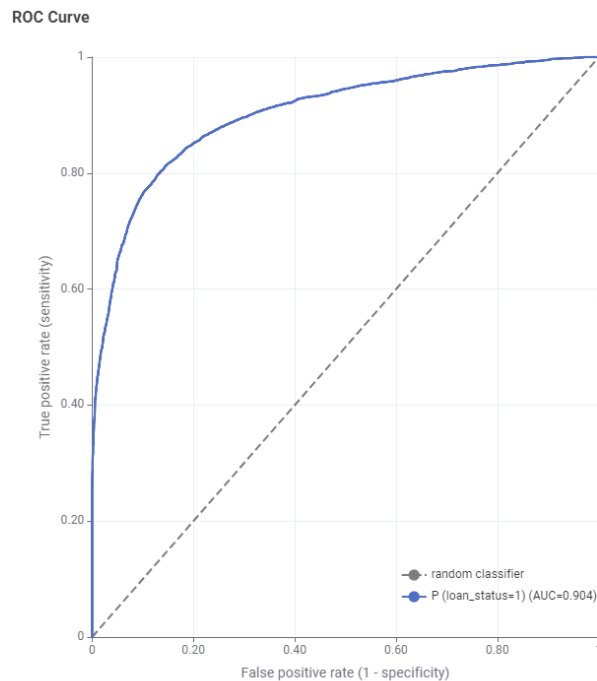


Figura 27. Nueva Curva ROC para Naive Bayes optimizado.

Alejandro Coman Venceslá

1.4.3 Árbol de Decisión

En este escenario vamos a comparar los rendimientos entre dos parámetros configurables en el nodo del Árbol de Decisión. Estos son “Quality measure” y “Pruning method”.

Quality measure nos dice qué criterio se utiliza para evaluar la calidad de una división en cada nodo del árbol. Este criterio determina cuál variable y punto de corte proporcionan la mejor separación de las clases en un nodo específico. Tenemos las opciones “Gain ratio” y “Gini index”.

Pruning method es la técnica utilizada para reducir el tamaño del árbol de decisión después de que ha sido construido. La poda ayuda a simplificar el árbol, eliminar nodos irrelevantes y reducir el riesgo de sobreajuste. Podemos no aplicar poda o una poda denominada “MDL”.

Como ya vimos en la configuración por defecto de este nodo en la Figura 19. Quality measure está inicializada a “Gini Index” y Pruning Method a “No pruning”. No obstante, vamos a ver qué ocurre con el resto de opciones.

| PRECISIÓN | | | AUC | | |
|----------------------------------|-------|--------------|----------------------------------|-------|--------------|
| Quality Measure / Pruning Method | No | MDL | Quality Measure / Pruning Method | No | MDL |
| Gini Index | 0,876 | 0,911 | Gini Index | 0.869 | 0,907 |
| Gain Ratio | 0,878 | 0,906 | Gain Ratio | 0,859 | 0.899 |

Tabla 19. Resultados al modificar parámetros en el Árbol de Decisión.

Vistos los resultados de utilizar las medidas de calidad y técnicas de poda de la Tabla 19, la configuración por defecto no es la que proporciona los mejores resultados. Podemos apreciar que la configuración que utiliza el Índice de Gini y MDL para podar nos obtiene una precisión de 0.911 y un AUC de 0.907, lo cual es una buena mejora con respecto a los datos anteriores. El incremento principal de rendimiento es al utilizar la poda. Esto tiene bastante sentido ya que es un aspecto crucial a la hora de conseguir un árbol de decisión mejor.

En adición, vamos a aplicar también el submuestreo de la clase mayoritaria en este algoritmo y ver si mejora el rendimiento del clasificador:

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0.911 | 0.903 |
| AUC | 0.907 | 0.941 |

Tabla 20. Comparación del rendimiento del Árbol de Decisión, entre aplicar o no submuestreo a la clase mayoritaria.

En la Tabla 20. Podemos ver que sí mejora notablemente el rendimiento al tener un valor AUC significativamente superior.

Alejandro Coman Venceslá

Nuevos resultados:

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 4322 | 794 |
| <i>loan_status</i> = 0 | 198 | 4918 |

Tabla 21. Matriz de confusión del Árbol de Decisión optimizado.

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 4322 | 198 | 4918 | 794 | 0.845 | 0.956 | 0.846 | 0.961 | 0.897 | |
| 0 | 4918 | 794 | 4322 | 198 | 0.961 | 0.861 | 0.961 | 0.846 | 0.908 | |
| Overall | | | | | | | | | | 0.903 |

Tabla 22. Estadísticas de precisión de Árbol de Decisión optimizado.

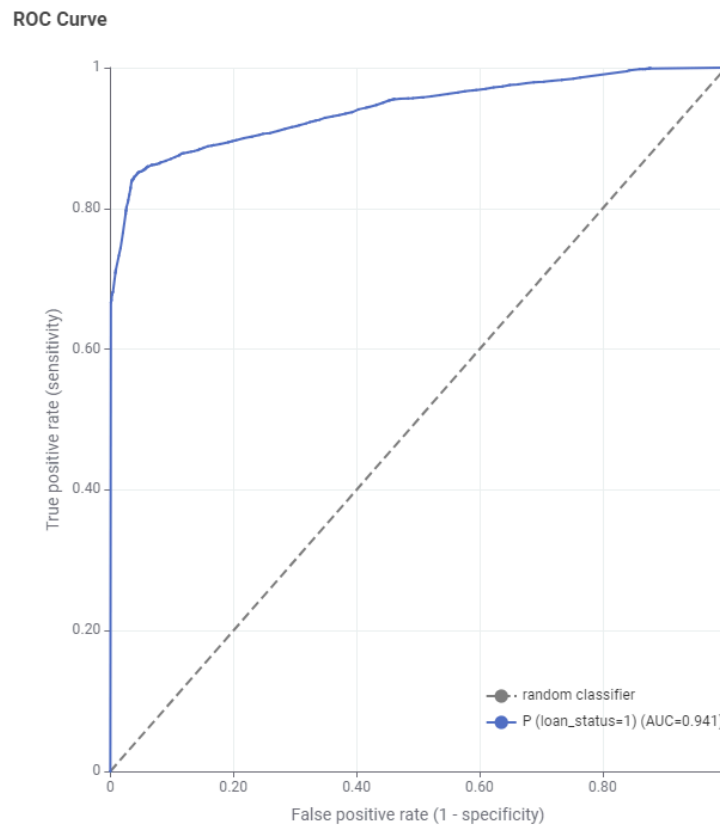


Figura 28. Nueva Curva ROC para el Árbol de Decisión Optimizado.

1.4.4 Gradient Boosted Trees

En este algoritmo vamos a configurar dos parámetros del nodo Gradient Boosted Trees. De manera similar al algoritmo de Random Forest. Tocaremos el parámetro de “Tree Depth”, el cual representa el límite de profundidad hasta el cual el árbol se puede expandir. Adicionalmente también retocaremos el parámetro “Number of models”, el cual se refiere a la cantidad de árboles individuales que se construirán y ensamblarán para crear el modelo final de boosting



Figura 29. Configuración por defecto del nodo Gradient Boosted Trees

Vamos a comprobar los cambios en el rendimiento que se darán al ir cambiando estos parámetros en el algoritmo.

| PRECISIÓN | | | | AUC | | | |
|----------------------------------|-------|-------|-------------|----------------------------------|-------|-------|-------------|
| Tree depth / Number of models | 50 | 100 | 200 | Tree depth / Number of models | 50 | 100 | 200 |
| 4 | 0,914 | 0,919 | 0,92 | 4 | 0,935 | 0,946 | 0,95 |
| 10 | 0,917 | 0,918 | 0,912 | 10 | 0,942 | 0,943 | 0,934 |
| 15 | 0,909 | 0,912 | 0,915 | 15 | 0,924 | 0,934 | 0,939 |

Tabla 23. Resultados al modificar parámetros en Gradient Boosted Trees.

Como podemos ver en los resultados de la tabla anterior, la mejor configuración de todas es aquella que tiene un límite de profundidad de 4 y número de modelos con valor de 200. Es mejor tener un límite más bajo ya que los Gradient Boosted Trees son muy potentes. Si creáramos árboles más profundos podríamos estar en un caso de sobreajuste.

Por parte del número de modelos, cada árbol se construye para corregir los errores del árbol anterior. Con más iteraciones (más árboles), el modelo puede refinar las predicciones con más precisión, ya que cada árbol contribuye con pequeñas mejoras. En adición, vamos a aplicar también el submuestreo de la clase mayoritaria en este algoritmo y ver si mejora el rendimiento del clasificador:

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0.92 | 0.924 |
| AUC | 0.95 | 0.971 |

Tabla 24. Comparación de rendimiento de Gradient Boosted Trees, al aplicar submuestreo.

Alejandro Coman Venceslá

Podemos apreciar que obtenemos un valor superior para la precisión y el AUC, con 0.971 este último. Un valor muy bueno.

Nuevos resultados:

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 4520 | 596 |
| <i>loan_status</i> = 0 | 185 | 4931 |

Tabla 25. Matriz de confusión de Gradient Boosted Trees optimizado.

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 4520 | 185 | 4931 | 596 | 0.8847 | 0.961 | 0.884 | 0.965 | 0.920 | |
| 0 | 4931 | 596 | 4520 | 185 | 0.964 | 0.892 | 0.965 | 0.884 | 0.927 | |
| Overall | | | | | | | | | | 0.924 |

Tabla 26. Estadísticas de precisión de Gradient Boosted Trees optimizado.

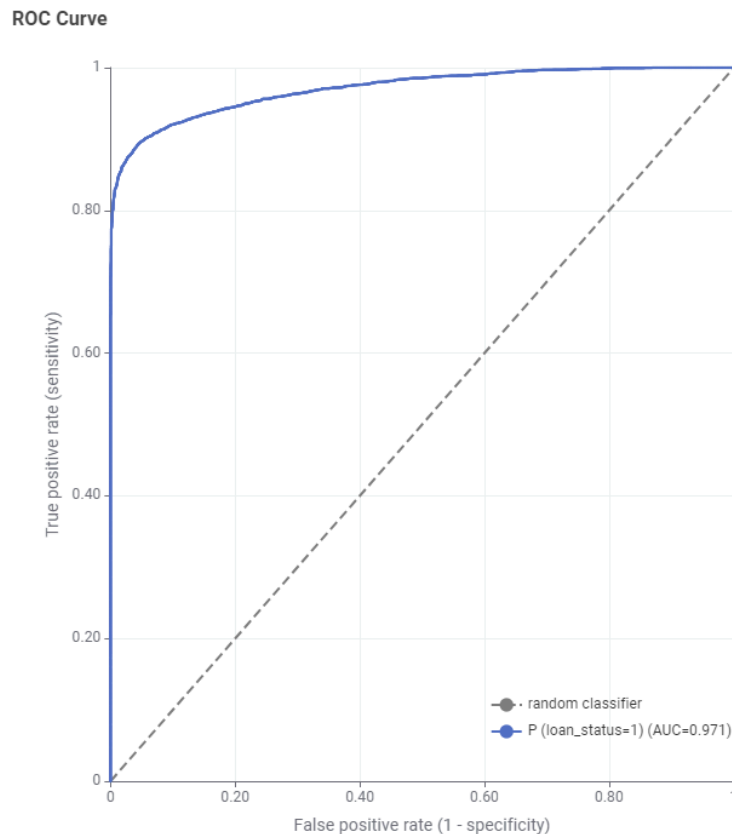


Figura 30. Nueva Curva ROC para el Gradient Boosted Trees Optimizado.

1.4.5 Stochastic Gradient Descent

Anteriormente hemos visto que este algoritmo es el que peor rendimiento ha presentado para la tarea de clasificación. Es por ello que es importante modificar los parámetros de este modelo. Vamos a utilizar funciones de pérdida (Loss functions) y ratios de aprendizaje (learning rate) distintos.

Las funciones de pérdida nos permiten calcular el error que tenemos en cada iteración de la red neuronal con respecto a los valores verdaderos. Este error se usa para ajustar en mayor o menor medida los parámetros de la red neuronal, hasta llegar a un punto de convergencia en el que el error sea lo suficientemente pequeño. El ratio de aprendizaje nos indica cuánto actualizaremos los parámetros de la red neuronal, según el error de la propia iteración.

Como pudimos apreciar en la Figura 20. Configuración por defecto del nodo, el “learning rate” es de 0.01 y la función de pérdida es Hinge Loss. Vamos a comparar las funciones de pérdida “Hinge loss” y “Log Loss”. Probaremos también un ratio de aprendizaje de 0.1, 0.01 y 0.001.

| PRECISIÓN | | | | AUC | | | |
|-------------------------------|-------|--------------|-------|-------------------------------|-------|--------------|-------|
| Loss Function / Learning Rate | 0.1 | 0.01 | 0.001 | Loss Function / Learning Rate | 0.1 | 0.01 | 0.001 |
| Log Loss | 0,842 | 0,849 | 0,842 | Log Loss | 0,86 | 0,868 | 0,86 |
| Hinge Loss | 0,849 | 0,849 | 0,84 | Hinge Loss | 0,773 | 0,769 | 0,755 |

Tabla 27. Resultados al modificar parámetros en Stochastic Gradient Descent.

Podemos observar que la configuración por defecto no es la más óptima. Log Loss es mejor en todos los casos estudiados. Además el ratio de aprendizaje óptimo es 0,01. Es importante escoger un learning rate que no sea muy bajo (ya que el modelo no va a converger tan rápido) ni muy alto (dar saltos muy grandes puede provocar que nos saltemos valores óptimos en el proceso de optimización, no llegando nunca a converger el modelo, incluso pudiendo llegar a divergir (el error se hace cada vez más grande)).

En adición, vamos a aplicar también el submuestreo de la clase mayoritaria en este algoritmo y ver si mejora el rendimiento del clasificador:

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0.849 | 0.843 |
| AUC | 0.868 | 0.907 |

Tabla 28. Comparación de rendimiento de SGD, al aplicar submuestreo.

Podemos ver que el valor de AUC mejora al aplicar submuestreo, volviendo a comprobar que esta técnica para solucionar el problema del desbalanceo de clases funciona.

Alejandro Coman Venceslá

Nuevos resultados:

| Clase / Predicción | <i>loan_status</i> (predicción) = 1 | <i>loan_status</i> (predicción) = 0 |
|------------------------|-------------------------------------|-------------------------------------|
| <i>loan_status</i> = 1 | 4171 | 945 |
| <i>loan_status</i> = 0 | 660 | 4456 |

Tabla 29. Matriz de confusión de SGD optimizado..

| <i>loan_status</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 4171 | 660 | 4456 | 945 | 0.815 | 0.863 | 0.815 | 0.871 | 0.839 | |
| 0 | 4456 | 945 | 4171 | 660 | 0.871 | 0.825 | 0.871 | 0.815 | 0.847 | |
| Overall | | | | | | | | | | 0.843 |

Tabla 30. Estadísticas de precisión de SGD optimizado.

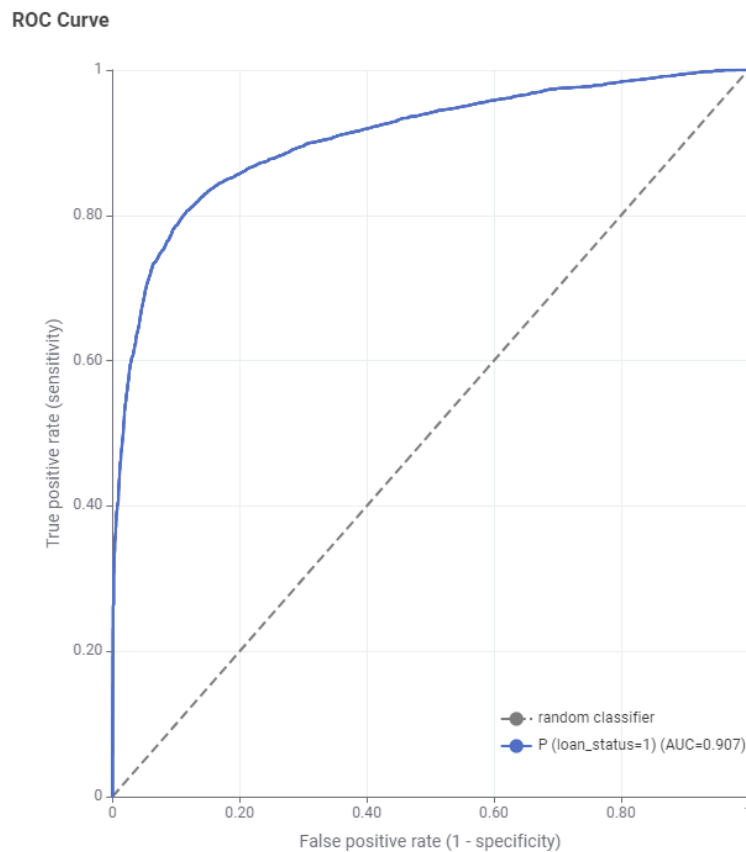


Figura 31. Nueva Curva ROC para el SGD optimizado.

Alejandro Coman Venceslá

Una vez hemos terminado de aplicar optimizaciones a todos los algoritmos vamos a ver en una sola gráfica como han quedado las nuevas curvas ROC, comparándolas con las antiguas.

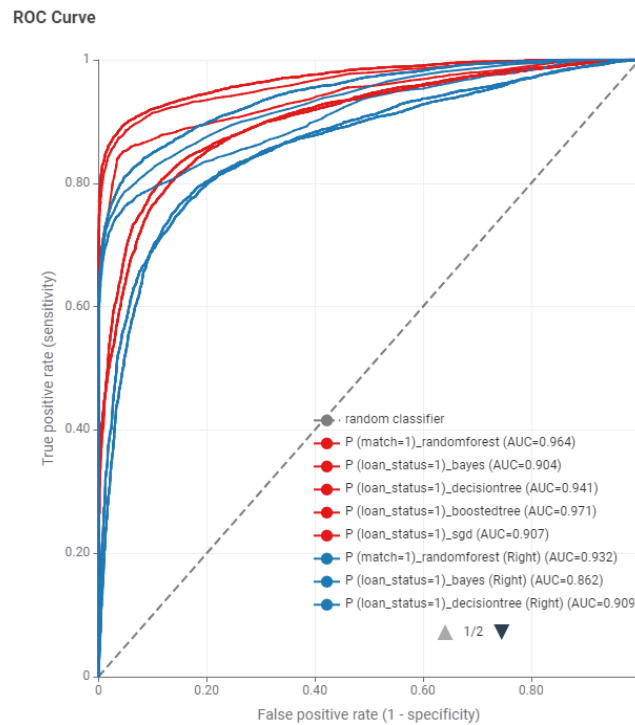


Figura 32. Comparación de curvas ROC entre los 5 algoritmos, optimizados y sin optimizar.

Como podemos ver en la Figura 32, donde comparamos las curvas ROC de los algoritmos sin optimizar y con optimizaciones (las curvas en azul son sin optimizaciones y en rojo con optimizaciones), nuestras mejoras han podido incrementar de manera notable el rendimiento de los algoritmos, ya que las curvas ROC nuevas se encuentran por encima de las de los algoritmos sin optimizar.

1.5 Análisis de resultados

Procedemos ahora a mostrar una tabla resumen con todos los algoritmos. Aquí incluiremos la tabla estadística del nodo Scorer y el valor AUC para cada algoritmo.

| | TP | FP | TN | FN | TPR | TNR | Accuracy | AUC |
|-------------------------------|------|-----|-------|------|-------|-------|----------|-------|
| Random Forest | 3795 | 305 | 13719 | 1321 | 0.741 | 0.978 | 0.918 | 0.937 |
| Naive Bayes | 3655 | 249 | 13775 | 1461 | 0.714 | 0.982 | 0.839 | 0.859 |
| Decision Tree | 3655 | 249 | 13775 | 1461 | 0.714 | 0.982 | 0.911 | 0.907 |
| Gradient Boosted Trees | 3856 | 262 | 13762 | 1260 | 0.754 | 0.981 | 0.92 | 0.95 |
| SGD | 3078 | 851 | 13173 | 2038 | 0.602 | 0.939 | 0.849 | 0.868 |

Tabla 31. Datos estadísticos de los 5 algoritmos.

Comparación entre algoritmos. TPR, TNR, Accuracy y AUC

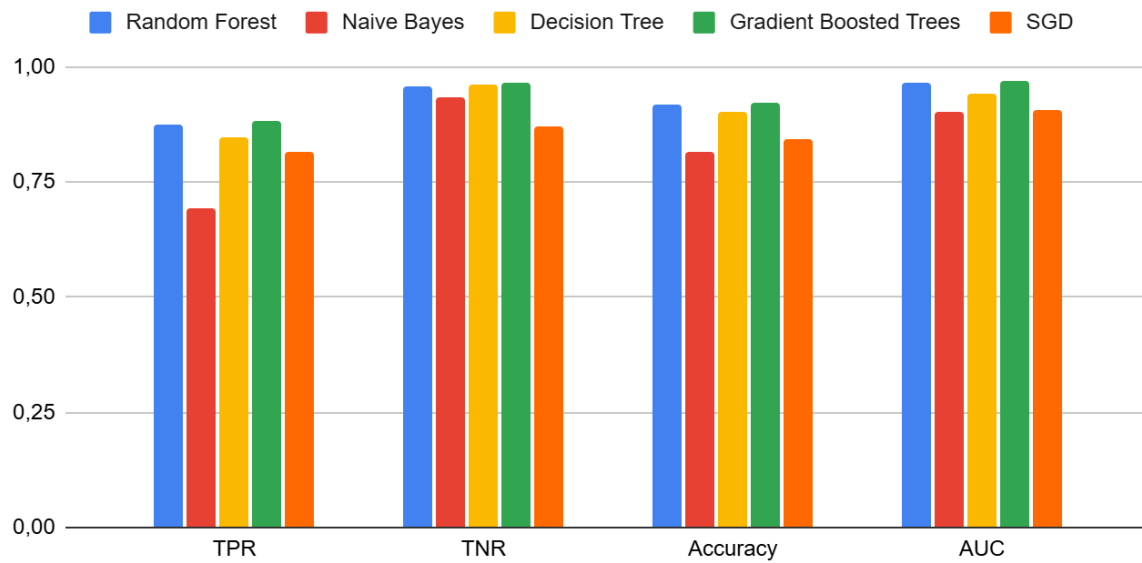


Figura 33. Gráfica comparativa de los 5 algoritmos, basada en la tabla 31.

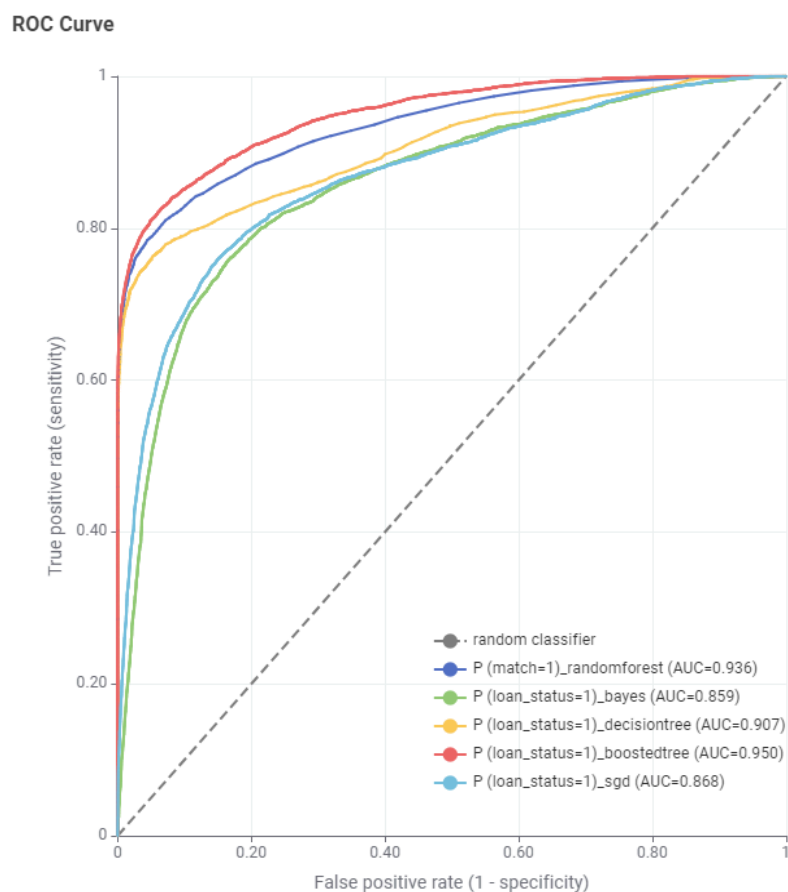


Figura 34. Curvas ROC de los cinco algoritmos.

Alejandro Coman Venceslá

Como podemos observar en los datos expuestos y en la gráfica y curvas ROC construidas a partir de los datos anteriores, en las Figuras 33 y 34, tenemos claro lo siguiente:

Gradient Boosted Trees y **Random Forest** destacan en el conjunto de datos de aprobación de préstamos debido a que son métodos de ensamble, propensos a proporcionar mejores resultados. Estos modelos de ensamble funcionan combinando varios árboles de decisión, lo que les permite capturar patrones no lineales y adaptarse a interacciones sutiles entre variables. Este aspecto es fundamental en un contexto donde los datos incluyen variables personales y financieras, que influyen conjuntamente en la decisión de aprobar o rechazar un préstamo. +

El árbol de decisión proporciona resultados algo inferiores a los dos algoritmos mencionados, pero mejores que SGD y Naive Bayes, ya que no es un modelo de ensamble. No obstante, al haber tratado las clases desbalanceadas, nos proporciona un rendimiento bastante bueno.

Finalmente, **SGD** y **Naive Bayes** muestran un rendimiento inferior debido a sus limitaciones estructurales. **SGD**, que es una técnica de optimización que se adapta bien a modelos lineales, no captura de manera adecuada las relaciones complejas que pueden existir entre variables de ingresos, empleo, historial crediticio, y otros factores, además tiende a sobre ajustarse más de la cuenta. **Naive Bayes**, por su parte, asume independencia condicional entre variables, lo cual es una suposición poco realista en datos financieros y personales donde existe una alta probabilidad de que las variables estén correlacionadas. Además, ambos algoritmos requieren un preprocesamiento más exhaustivo para manejar variables categóricas y valores perdidos, lo que limita su efectividad frente a modelos más complejos como Gradient Boosted Trees y Random Forest.

1.6 Interpretación de los datos

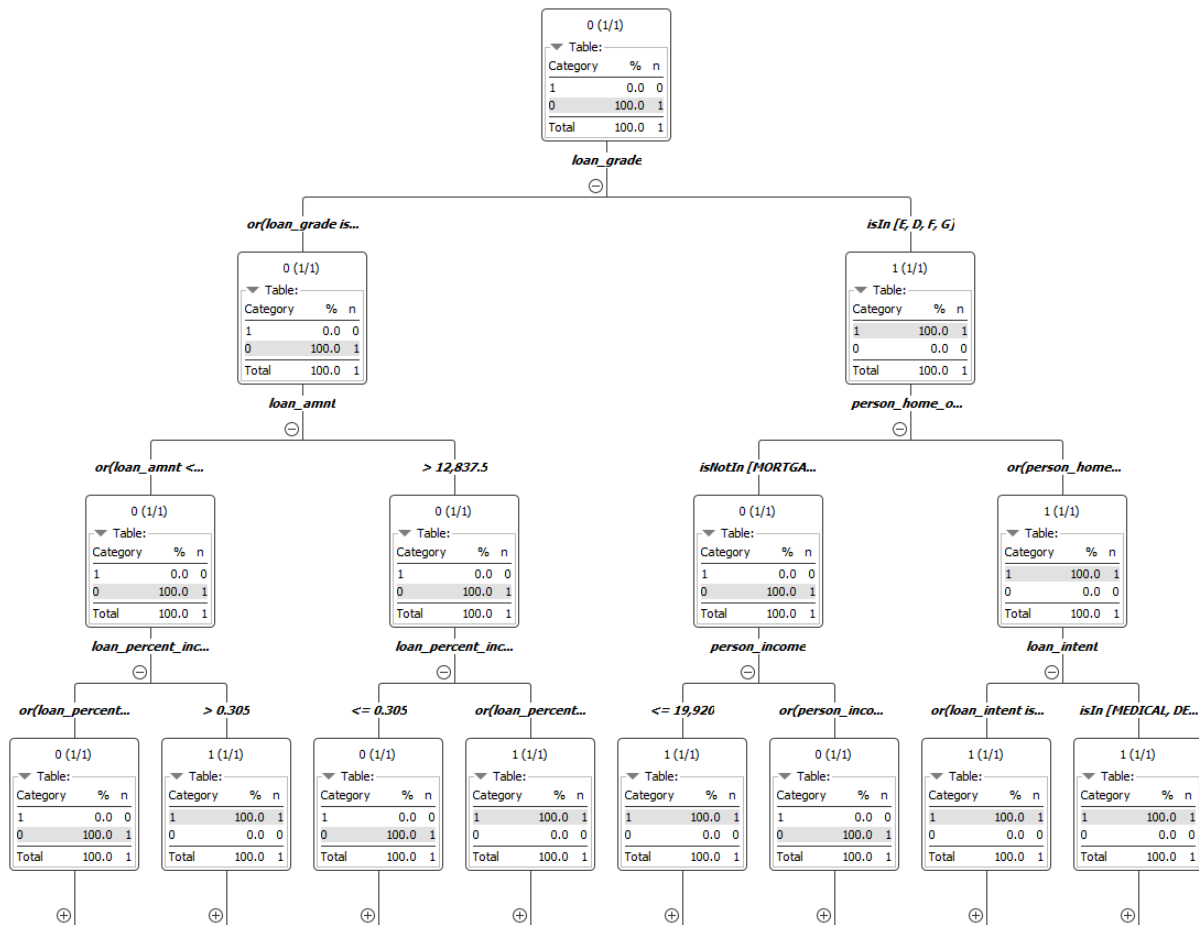


Figura 30. Árbol de decisión (modelo 1) de Random Forest

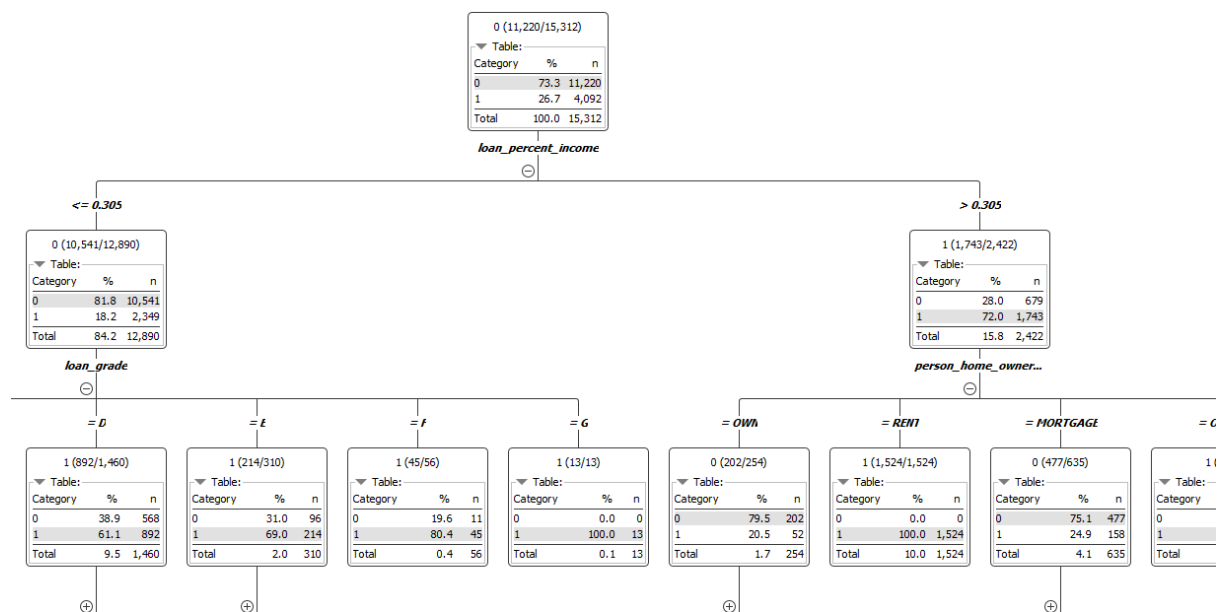


Figura 31. Árbol de decisión de Decisión Tree.

Alejandro Coman Venceslá

En el Random Forest vemos que la característica que mejor separa las clases es “**loan_grade**”. Ya va a empezar a clasificar en dos grandes grupos, según si el grado de riesgo es mejor que D (incluido) o peor que E (incluido). En el caso de que el crédito tenga poco riesgo, comienza a analizar la cantidad del préstamo. En caso de que el crédito tenga un riesgo elevado, lo que va a hacer es comprobar la situación de la casa del solicitante.

Por otro lado, en Decision Tree vemos que la característica que mejor separa las clases es “**loan_percent_income**”. Parece ser que el árbol de decisión simple prefiere enfocarse en qué porcentaje de la renta del solicitante representa el crédito solicitado. Si el porcentaje es bajo, mira el nivel de riesgo del crédito solicitado. En el caso de que el porcentaje sea alto, comprueba la situación de la casa del solicitante.

No obstante, estos dos árboles parecen seguir una estructura similar, se enfocan en tres aspectos clave de los datos:

1. La cantidad que el solicitante va pedir, ya sea en términos absolutos o en el porcentaje que representa el crédito con respecto a la renta
2. El nivel de riesgo que tiene el crédito pedido
3. La situación de la casa del solicitante (si alquila su casa o si la ha comprado con una hipoteca).

Podemos concluir que quizá los atributos como la longitud del historial crediticio, si el solicitante ha resultado en impagos anteriormente o la edad del solicitante no son tan importantes para realizar decisiones clave en la tarea de clasificación.

1.7 Contenido adicional

1.8 Bibliografía

- <https://ccia.ugr.es/~casillas/knime.html>
- https://hub.knime.com/knime/spaces/Examples/02_ETL_Data_Manipulation/01_Filtering/03_Row_Filtering~zzjm_YTvol2flBdP/most-recent
- <https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.preproc.correlation.filter.CorrelationFilterNodeFactory>
- https://hub.knime.com/knime/extensions/org.knime.features.ext.weka_3.7/latest
- https://hub.knime.com/knime/extensions/org.knime.features.ext.weka_3.7/latest/org.knime.ext.weka37.classifier.WekaClassifierNodeFactory:9a6242d3
- <https://hub.knime.com/vittoriohaardt/spaces/Public/Cross%20Validation%20example~f3DLUOsU8xclZBjc/current-state>
- <https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.preproc.normalize3.Normalizer3NodeFactory>
- <https://elvex.ugr.es/decsai/intelligent/workbook/d1%20knime.pdf>
- https://docs.knime.com/2018-12/analytics_platform_quickstart_guide/index.html#start-knime-analytics-platform
- Prado de la Asignatura

2. Segunda Cita

2.1. Introducción

Este conjunto de datos contiene datos recogidos de eventos de citas rápidas experimentales, en los que los participantes tuvieron citas de cuatro minutos con cada persona del sexo opuesto. Al final de cada cita, se les preguntaba si les gustaría volver a ver a su cita, registrando así su interés en una segunda cita. Además, los participantes puntuaron a sus citas en seis atributos clave: atractivo, sinceridad, inteligencia, diversión, ambición e intereses compartidos.

Además de las respuestas directas de cada cita, el dataset incluye información más detallada de cada participante, obtenida a través de cuestionarios. Estos cuestionarios capturan datos demográficos (como edad y origen étnico), hábitos de citas, autopercepción de sus propios atributos y creencias sobre lo que consideran valioso en una pareja. También se recopila información sobre el estilo de vida, lo que ofrece un contexto más amplio sobre la personalidad y preferencias de cada individuo.

Consideraciones sobre el Dataset:

1. **Variedad en atributos:** es rico en atributos que cubren tanto aspectos objetivos (como demografía y puntuaciones dadas a cada cita) como subjetivos (autopercepción y creencias personales), proporcionando una base compleja y variada para la predicción de la intención de una segunda cita.
2. **Hay un desbalance de clases:** ya que hay muchas más instancias con segundas citas rechazadas que aceptadas. Será necesario encargarnos de este desbalance posteriormente. Tenemos exactamente 6998 ocurrencias rechazadas y 1380 aceptadas.
3. **Valores faltantes:** gran parte de los atributos presentan valores faltantes. Necesitaremos aplicar alguna técnica de imputación o de eliminación más adelante.

Para realizar la predicción de este problema he seleccionado los siguientes algoritmos:

1. **Random Forest:** Al combinar múltiples árboles de decisión, son robustos y presentan un buen rendimiento al trabajar con conjuntos de datos mixtos como el de este problema.
2. **Naive Bayes:** Este algoritmo es bastante rápido y eficiente, lo cual viene bien, contando con que tenemos una gran cantidad de instancias. Además, también maneja bien los conjuntos de datos mixtos.
3. **XGBoost:** ideal para este problema de clasificación porque maneja bien conjuntos de datos con atributos mixtos, complejidad y desbalance de clases, capturando relaciones no lineales y patrones sutiles en los datos de preferencias y percepciones personales.
4. **Gradient Boosted Trees:** Método de ensamble que construye secuencialmente árboles de decisión, donde cada nuevo árbol intenta corregir los errores del anterior.

Alejandro Coman Venceslá

Es conocido por su alto rendimiento predictivo y su capacidad para capturar relaciones no lineales y patrones complejos en los datos. En el dataset de aprobación de créditos, puede manejar eficientemente variables mixtas y el desbalance de clases.

5. **SGD:** SGD es un algoritmo de optimización ampliamente utilizado en el aprendizaje automático y la clasificación. Presenta un buen rendimiento en problemas de clasificación binaria.

2.2. Procesado de datos

En este conjunto de datos no tenemos la presencia de valores extremos ya que la mayoría de datos indican preferencias del 0 al 10 sobre una cualidad en específico. No obstante para el tema de los valores faltantes, ya que la cantidad de instancias es bastante alta y el número de valores faltantes tampoco es demasiado alto, vamos a rellenar los huecos de la siguiente manera, con el nodo “Missing Value”

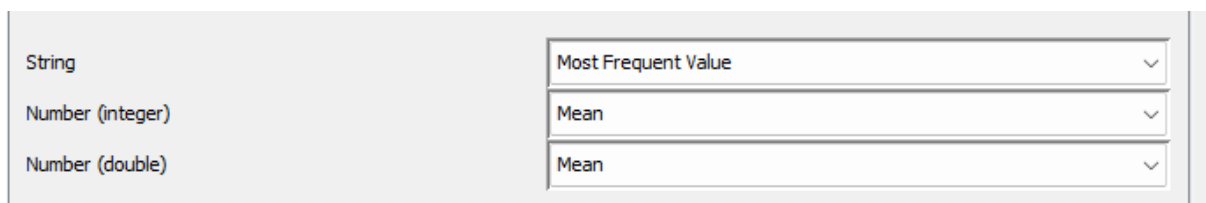


Figura 1. Configuración del nodo Missing Value para el preprocesado de los datos.

Una vez tratado con los valores faltantes, he realizado un estudio de correlación lineal de cada variable, con respecto a la variable objeto “match”, ya que tenemos muchas variables en el modelo. Esto lo he hecho de la siguiente manera:

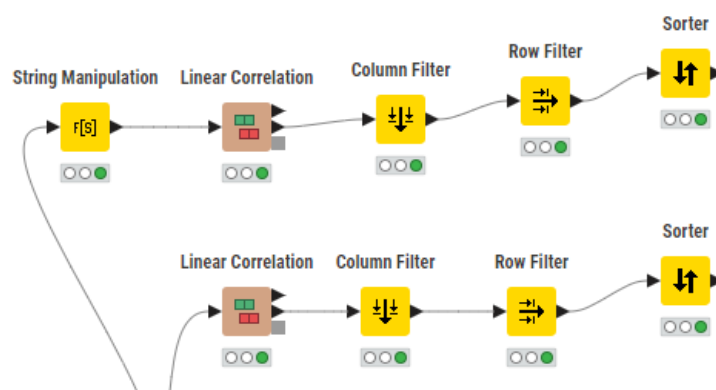


Figura 2. Análisis de correlación lineal en el preprocesado general de los datos.

He realizado uno para las variables categóricas y otro para las variables numéricas (convirtiendo respectivamente la variable objeto “match”). Al obtener la matriz de correlación, me he quedado solamente con la columna de correlaciones de “match”. Posteriormente, he dejado fuera las variables que tuvieran una correlación menor a 0.05 mediante el nodo “Row Filter”:

Criterion 1

Filter column: match

Operator: Greater than or equal

Value: 0.05

Figura 3. Nodo Row Filter, para filtrar las variables con poca correlación con la variable “match”.

Finalmente he ordenado las variables con el nodo “Sorter” para ver la lista de correlaciones ordenada y así es como he realizado el estudio de correlación, quedándome con una lista de variables que tienen una correlación superior a 0.05. Sabiendo esto, he seleccionado manualmente en un “Column Filter” dichas variables para que sean incluidas en el modelo y he dejado el resto fuera.

Excludes:

- gender
- age
- age_o
- d_age
- samerace
- importance_same_race
- Any unknown column

Includes:

- d_d_age
- race
- race_o
- d_importance_same_race
- attractive_o
- sinsere_o
- intelligence_o

Figura 4. Column Filter para filtrar variables poco correladas fuera del modelo.

Finalmente, he normalizado todas las variables. El nodo de preprocesamiento general para este conjunto de datos queda de la siguiente manera:

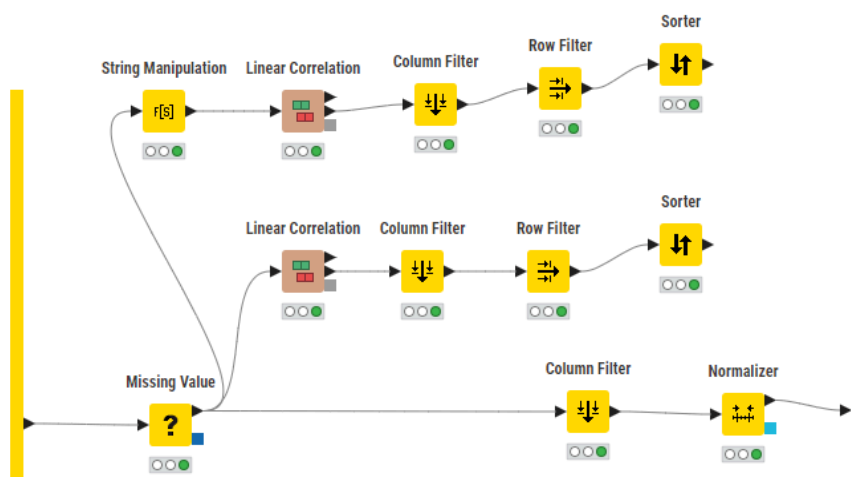


Figura 5. Flujo de datos para el preprocesamiento general.

Alejandro Coman Venceslá

El resto del flujo de datos se estructura de la siguiente manera, muy similar al primer dataset de aprobación de créditos:

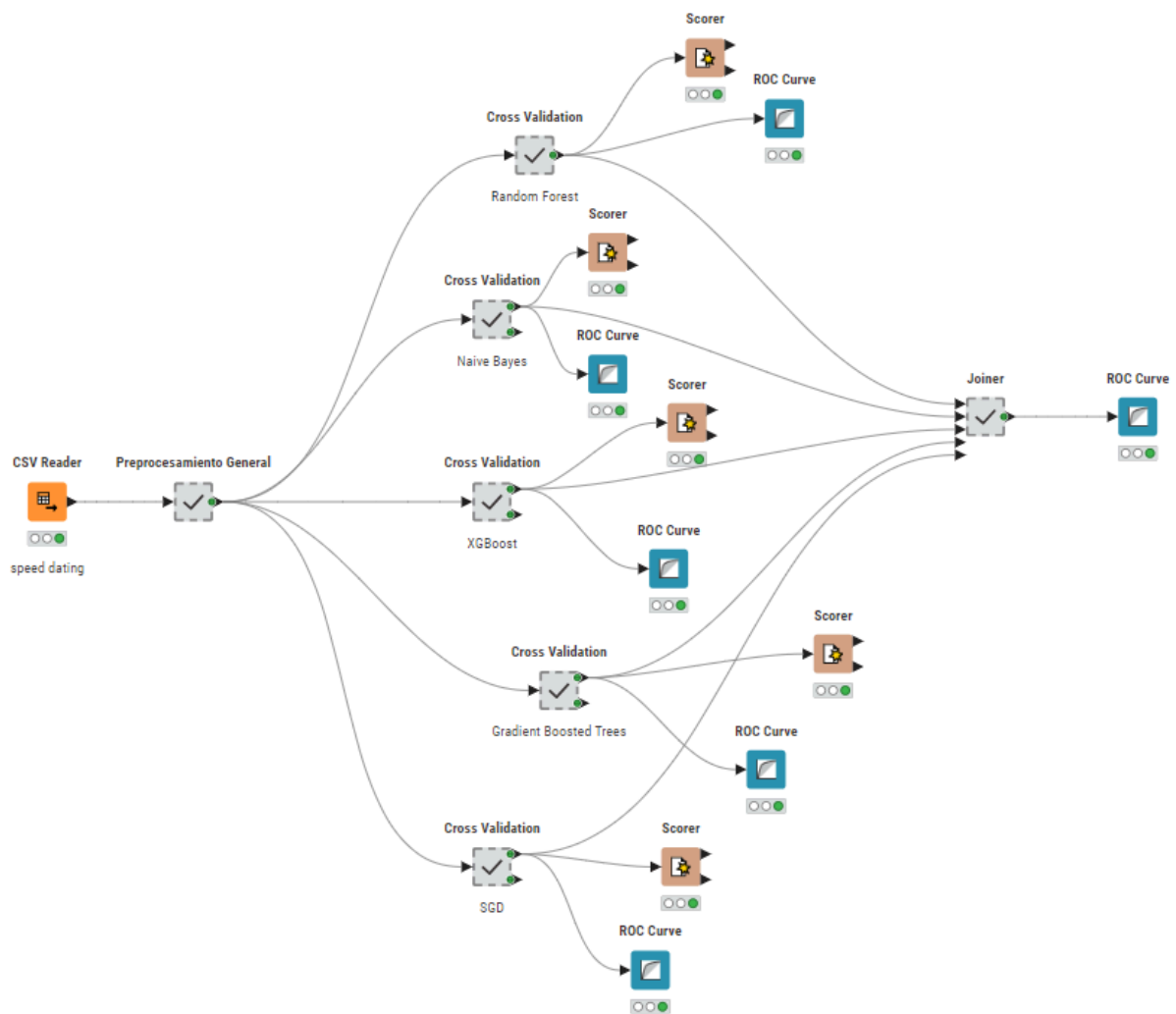


Figura 6. Vista general del flujo de trabajo para el problema de la segunda cita.

Tras leer los datos del archivo CSV y aplicar el preprocesamiento anteriormente mencionado, mandaremos estos datos a cada algoritmo, el cual tendrá una validación cruzada con 5 validaciones. Todos los nodos de validación cruzada (X-Partitioner y X-Aggregator) tienen la siguiente configuración:

| | |
|--|---|
| Target column | <input type="text" value="S match"/> |
| Prediction column | <input type="text" value="S Prediction (match)"/> |
| <input type="checkbox"/> Add column with fold id | |

| | |
|-----------------------|--------------------------------------|
| Number of validations | <input type="text" value="5"/> |
| Linear sampling | <input type="radio"/> |
| Random sampling | <input type="radio"/> |
| Stratified sampling | <input checked="" type="radio"/> |
| Class column | <input type="text" value="S match"/> |

Figuras 7 y 8. Configuración de los nodos X-Partitioner (izquierda) y X-Aggregator (derecha) para la validación cruzada.

Alejandro Coman Venceslá

En las Figuras 7 y 8 muestro la configuración de los nodos encargados de la validación cruzada. Se ha utilizado “stratified sampling” basado en la clase objetivo (loan_status en este caso), para intentar aminorar el problema del desbalanceo de las clases.

Una vez terminan de ejecutarse todos los algoritmos, para poder ver los resultados, se usa para cada algoritmo los nodos **Scorer** y **ROC Curve**. En adición se ha construido otro metanodo para unir todas las columnas de las predicciones en una sola tabla y así poder ver todas las curvas ROC en un solo nodo. Esto lo podemos apreciar en la Figura 9.

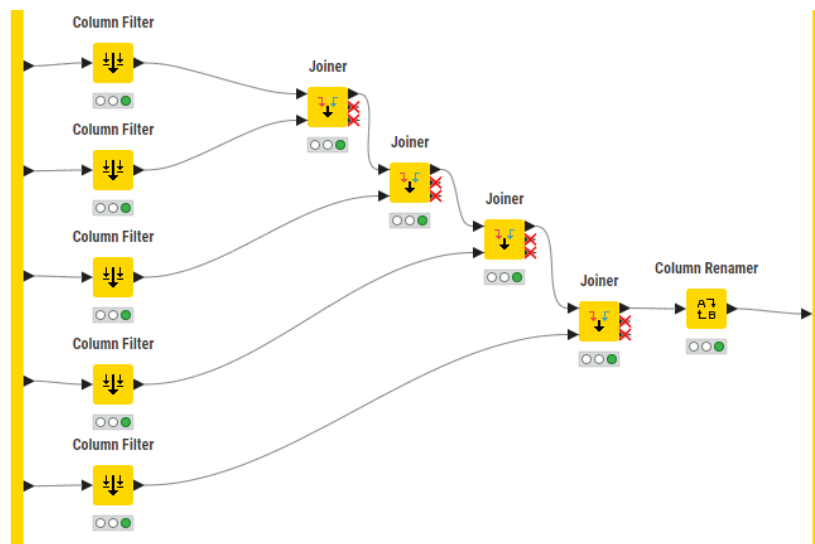


Figura 9. Metanodo para juntar las columnas de las predicciones de los distintos algoritmos y representarlas todas las curvas ROC juntas.

2.3 Resultados obtenidos

A continuación vamos a comprobar los resultados que hemos obtenido para cada uno de los algoritmos, habiendo aplicado este preprocesamiento general y habiendo dejado la configuración de los algoritmos por defecto. Mostraremos las matrices de confusión y las tablas de estadísticas de los nodos Scorer, además de las curvas ROC individuales. También podremos analizar las primeras métricas que obtenemos de cada algoritmo.

2.3.1 Random Forest

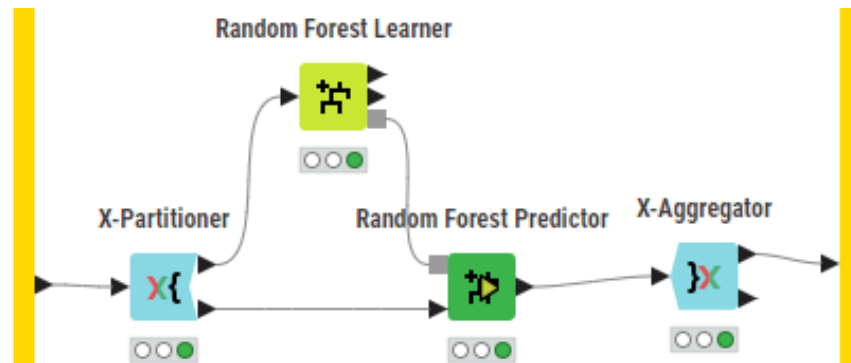


Figura 10. Metanodo de la validación cruzada para el algoritmo "Random Forest".

| Clase / Predicción | <i>match</i> (predicción) = 1 | <i>match</i> (predicción) = 0 |
|--------------------|-------------------------------|-------------------------------|
| <i>match</i> = 1 | 387 | 993 |
| <i>match</i> = 0 | 171 | 6687 |

Tabla 1. Matriz de confusión de Random Forest.

| <i>match</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 387 | 171 | 6827 | 993 | 0.280 | 0.6942 | 0.280 | 0.976 | 0.399 | |
| 0 | 6827 | 993 | 387 | 171 | 0.976 | 0.873 | 0.976 | 0.280 | 0.921 | |
| Overall | | | | | | | | | | 0.861 |

Tabla 2. Estadísticas de precisión de Random Forest.

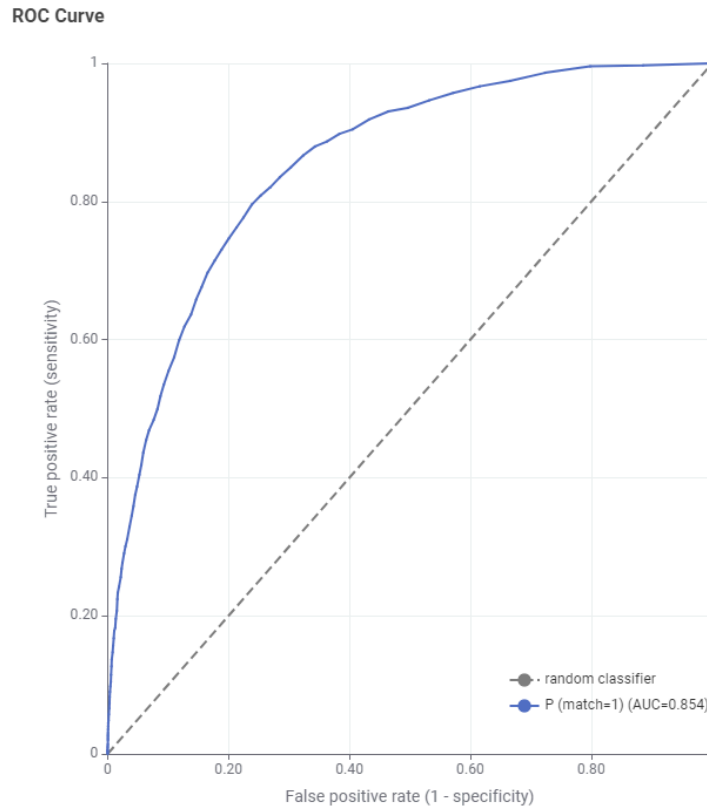


Figura 11. Curva ROC de Random Forest.

En la Figura 11 mostramos la curva ROC de las predicciones de este algoritmo. Además el AUC es de 0.854, lo cual indica un rendimiento bueno. Además, fijándonos en las Tablas 1 y 2, el Accuracy es de 0.861. Podemos volver a ver el fenómeno en el que tenemos una gran cantidad de falsos negativos, provocada por el desbalanceo de la clase “match”. No obstante, intentaremos mejorar el rendimiento más adelante.

2.3.2 Naive Bayes

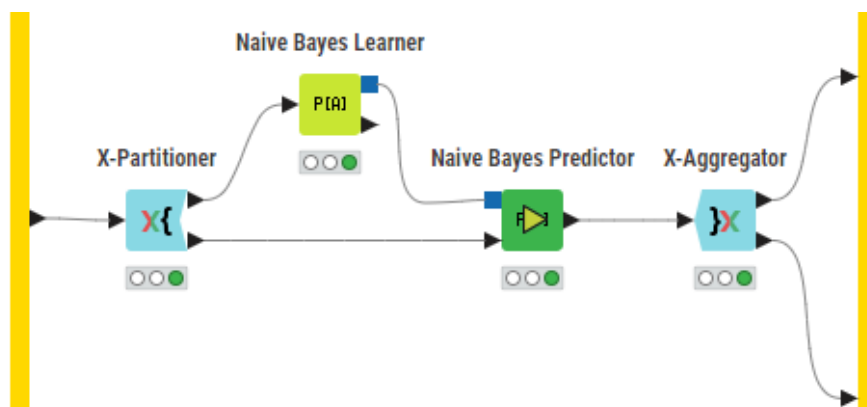


Figura 12. Metanodo de la validación cruzada para el algoritmo “Naive Bayes”.

| Clase / Predicción | <i>match</i> (predicción) = 1 | <i>match</i> (predicción) = 0 |
|--------------------|-------------------------------|-------------------------------|
| <i>match</i> = 1 | 5468 | 1530 |
| <i>match</i> = 0 | 376 | 1004 |

Tabla 3. Matriz de confusión de Naive Bayes.

| <i>match</i> | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|--------------|------|------|------|------|--------|-----------|-------------|-----------|-------|-------|
| 1 | 5468 | 376 | 1004 | 1530 | 0.781 | 0.936 | 0.781 | 0.728 | 0.852 | |
| 0 | 1004 | 1530 | 5468 | 376 | 0.728 | 0.396 | 0.728 | 0.781 | 0.513 | |
| Overall | | | | | | | | | | 0.772 |

Tabla 4. Estadísticas de precisión de Naive Bayes.

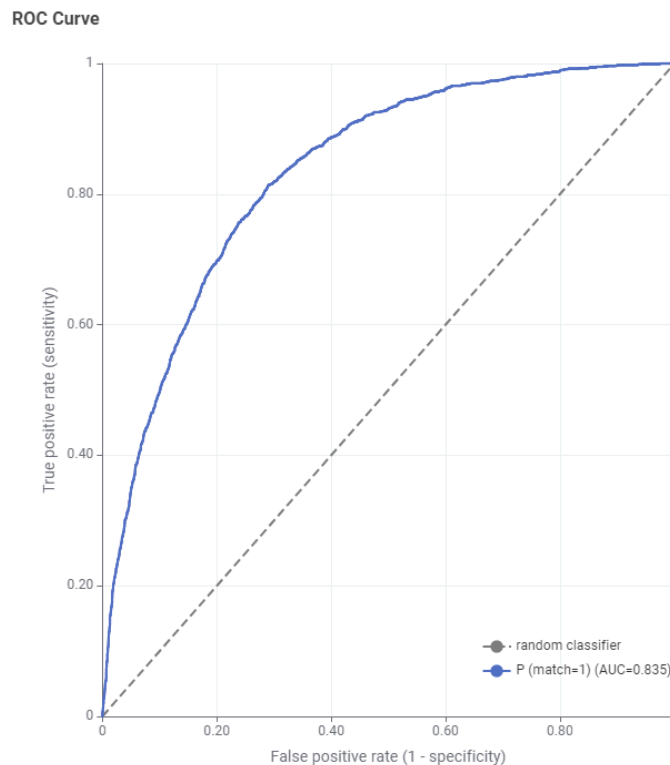


Figura 13. Curva ROC de Naive Bayes.

En este escenario, tras ver las Tablas 3 y 4 y la Figura 13, podemos observar como el algoritmo Naive Bayes tiene Accuracy algo más bajo (0.772) y un valor de AUC alto 0.835. De momento tenemos unas medidas ligeramente peores que Random Forest. Esto nos ocurría de igual manera en el primer set de datos de aprobación de créditos. Nos podemos remontar de nuevo, al hecho de que Naive Bayes asume independencia entre todas las variables del modelo. Esto provoca que las probabilidades de cada clase estén, a priori, sesgadas hacia la clase mayoritaria. Además aquí tenemos un número de variables bastante más alto. Veremos más adelante si podemos aminorar estas ineficiencias.

2.3.3 XGBoost

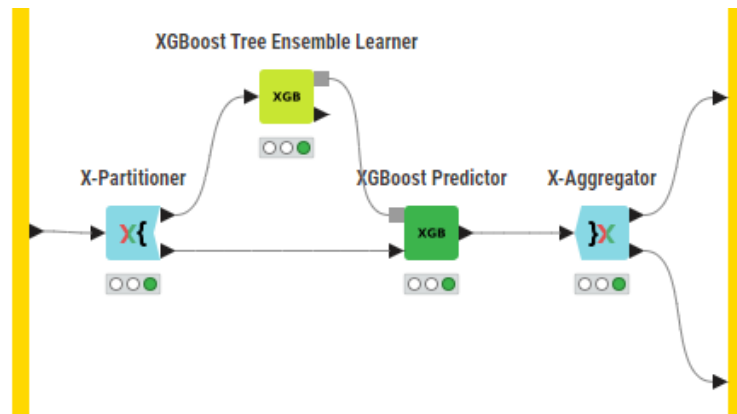


Figura 14. Metanodo de la validación cruzada para el algoritmo “XGBoost”.

Figura 15. Configuración por defecto del nodo “XGBoost”

| Clase / Predicción | <i>match</i> (predicción) = 1 | <i>match</i> (predicción) = 0 |
|--------------------|-------------------------------|-------------------------------|
| <i>match</i> = 1 | 536 | 844 |
| <i>match</i> = 0 | 384 | 6614 |

Tabla 5. Matriz de confusión de XGBoost.

| match | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|---------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 536 | 384 | 6614 | 844 | 0.388 | 0.583 | 0.388 | 0.945 | 0.466 | |
| 0 | 6614 | 844 | 536 | 384 | 0.945 | 0.887 | 0.945 | 0.388 | 0.915 | |
| Overall | | | | | | | | | | 0.853 |

Tabla 6. Estadísticas de precisión de XGBoost.

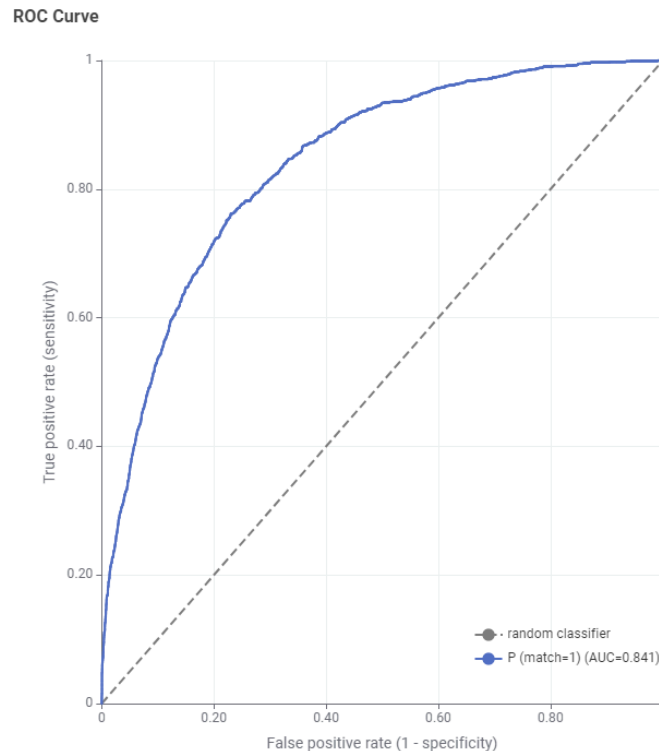


Figura 16. Curva ROC de XGBoost.

En este escenario, tras observar las Tablas 5 y 6 y las Figuras 15 y 16, podemos ver cómo el algoritmo XGBoost tiene también un Accuracy alto (0.853) y una AUC alta también, como se puede apreciar en la curva ROC (0.841). Son métricas que indican un buen rendimiento, pero aún mejorable. Son también peores que Random Forest, esto se puede deber al hecho de que XGBoost, al utilizar una técnica de boosting, se crean árboles menos profundos, que se optimizan a lo largo de las iteraciones del algoritmo. Random Forest suele crear árboles más profundos. Al tener un modelo con una gran cantidad de variables es posible que los árboles que XGBoost está creando no puedan aprender bien del todo las características de los datos. Es por ello que, intentaremos igualmente ajustar el algoritmo más adelante para comprobar si podemos mejorar las cifras.

2.3.4 Gradient Boosted Trees

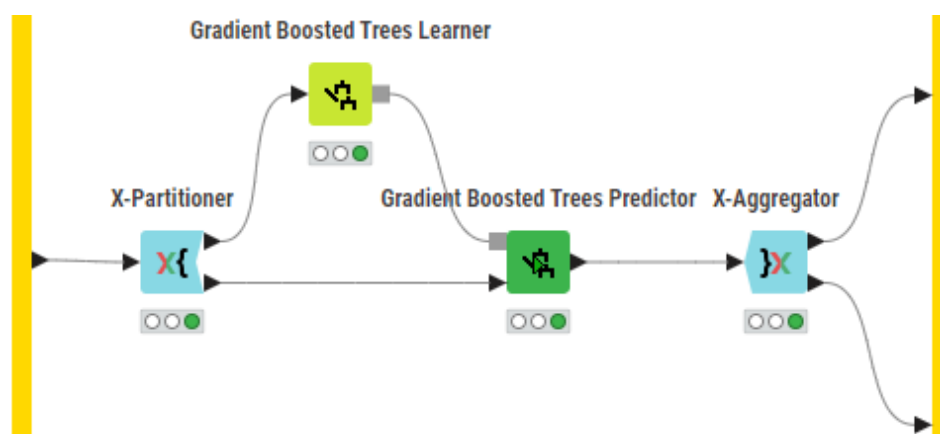


Figura 17. Metanodo de la validación cruzada para el algoritmo "Gradient Boosted Trees".

| Clase / Predicción | <i>match</i> (predicción) = 1 | <i>match</i> (predicción) = 0 |
|--------------------|-------------------------------|-------------------------------|
| <i>match</i> = 1 | 502 | 878 |
| <i>match</i> = 0 | 282 | 6716 |

Tabla 7. Matriz de confusión de Gradient Boosted Trees.

| match | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|---------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 502 | 282 | 6716 | 878 | 0.364 | 0.64 | 0.364 | 0.96 | 0.464 | |
| 0 | 6716 | 878 | 502 | 282 | 0.96 | 0.88 | 0.96 | 0.364 | 0.921 | |
| Overall | | | | | | | | | | 0.862 |

Tabla 8. Estadísticas de precisión de Gradient Boosted Trees.

ROC Curve

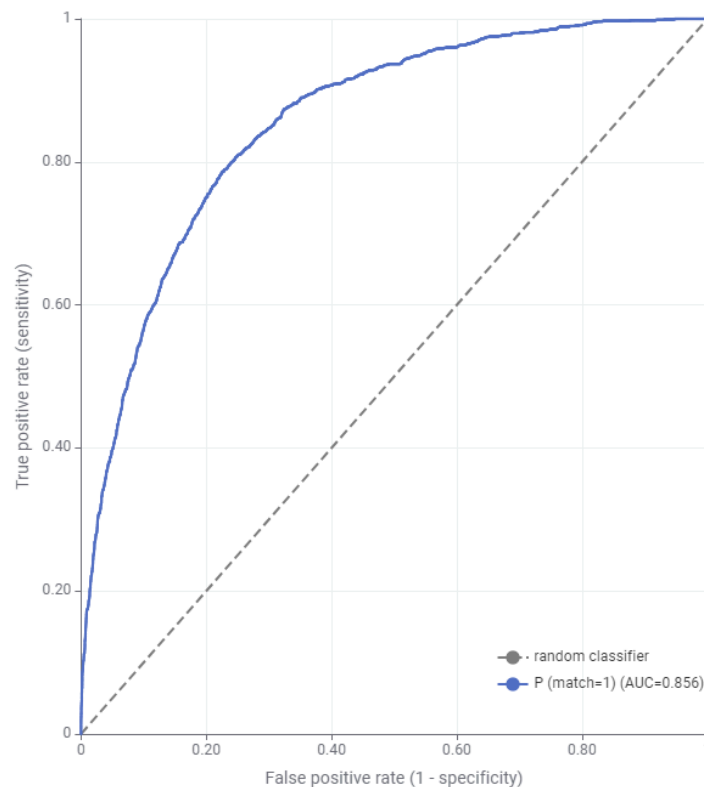


Figura 18. Curva ROC de Gradient Boosted Trees.

Como podemos observar en las Tablas 7 y 8 y en la Figura 18, hemos obtenido en este escenario una precisión con valor 0.862 y un área debajo de la curva (AUC) con valor 0.856, las cuales son algo mejores que XGBoost. Debemos recordar que XGBoost es una implementación supuestamente optimizada de Gradient Boosted Trees. No obstante, estamos probando nodos con sus valores por defecto de momento. Más adelante cuando hagamos algunos retoques a los modelos intentaremos mejorar los rendimientos de los clasificadores.

2.3.5 Stochastic Gradient Descent

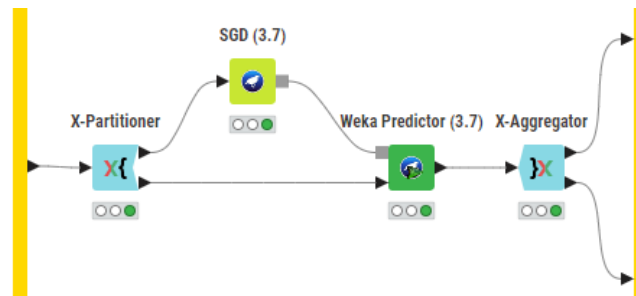


Figura 19. Metanodo de la validación cruzada para el algoritmo “Stochastic Gradient Descent”

| | |
|--------------|------------------|
| learningRate | 0.01 |
| lossFunction | Hinge loss (SVM) |

Figura 20. Configuración por defecto del nodo “SGD”

| Clase / Predicción | match (predicción) = 1 | match (predicción) = 0 |
|--------------------|------------------------|------------------------|
| match = 1 | 502 | 973 |
| match = 0 | 217 | 6781 |

Tabla 9. Matriz de confusión de SGD.

| match | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|---------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 407 | 217 | 6781 | 973 | 0.295 | 0.652 | 0.295 | 0.969 | 0.406 | |
| 0 | 6781 | 973 | 407 | 217 | 0.967 | 0.875 | 0.967 | 0.295 | 0.919 | |
| Overall | | | | | | | | | | 0.858 |

Tabla 10. Estadísticas de precisión de SGD.

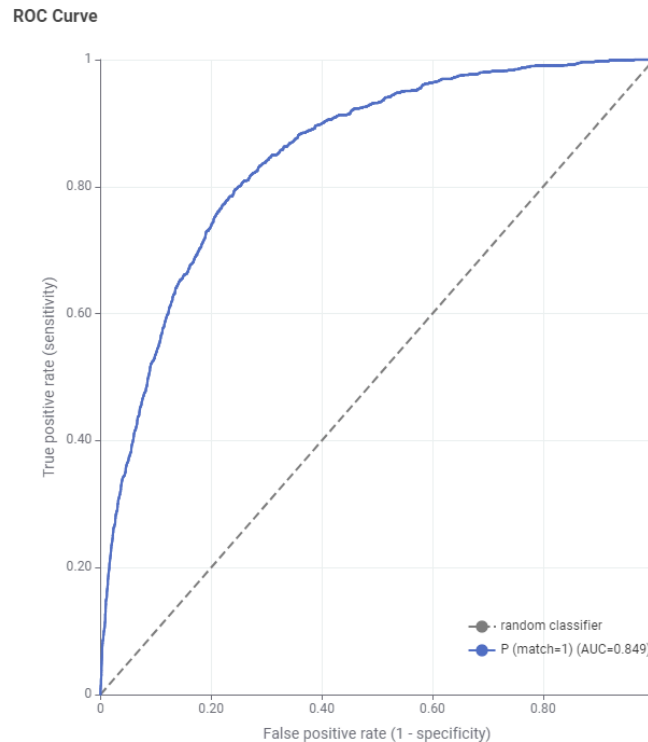


Figura 21. Curva ROC de SGD.

Visto lo expuesto en las Tablas 9 y 10 y la Figura 21, con un Accuracy de 0.858 y un AUC de 0.849. Vemos que obtenemos buenos resultados en la configuración por defecto del algoritmo. No obstante, es mejorable ya que SGD tiende a sobre ajustarse a los datos con facilidad. También vemos una alta cantidad de falsos negativos, debido al desbalanceo de la clase objeto. Intentaremos mejorar el rendimiento de este algoritmo más adelante.

Alejandro Coman Venceslá

2.3.6 Curvas ROC

A continuación mostramos todas las curvas ROC juntas.

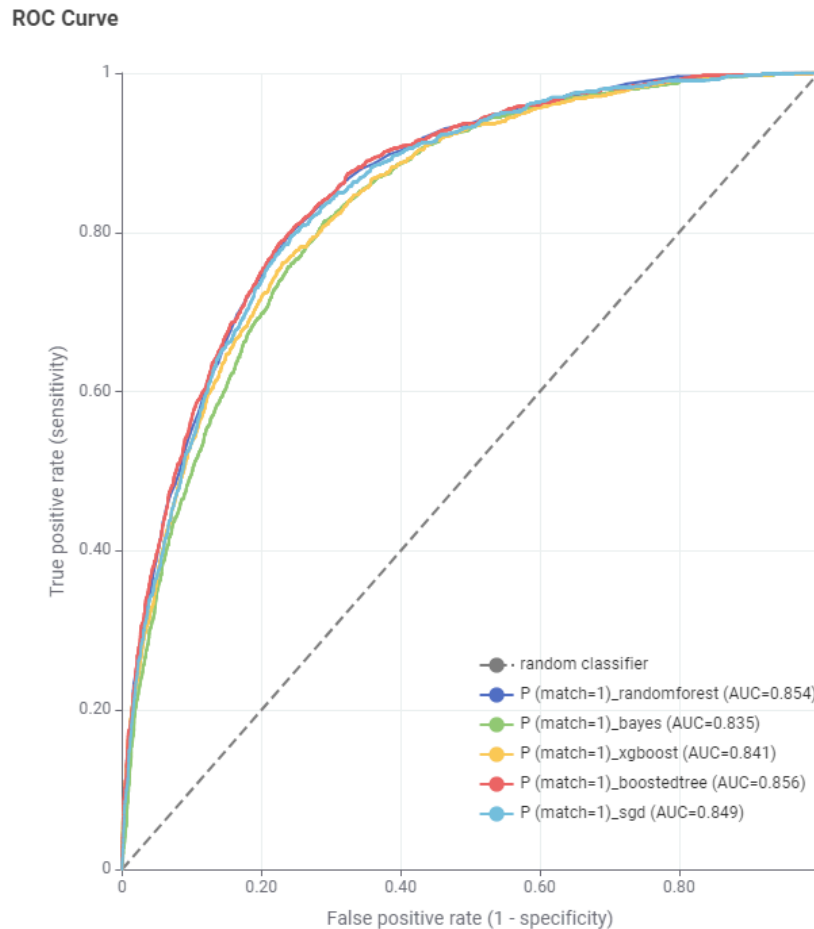


Figura 22. Curvas ROC de los 5 algoritmos.

En la Figura 22 mostramos las curvas ROC de los 5 algoritmos que se han empleado para este segundo conjunto de datos. En general, hemos propuesto unos algoritmos que dan bastante buen resultado. Todos presentan un rendimiento bastante similar con un área debajo de la curva con un valor aproximado de 0.85. El que peor rendimiento presenta es Naive Bayes, que se puede deber al hecho de la asunción de independencia entre variables, al haber una gran cantidad de variables es posible que esté perjudicando el rendimiento del modelo. El algoritmo que mejor rendimiento presenta es Gradient Boosted Trees, el cual es un modelo de ensamble, que genera muchos árboles de decisión los cuales van aprendiendo de los errores de los anteriormente generados.

En el siguiente apartado vamos a intentar corregir algunas ineficiencias de los algoritmos para comprobar si podemos mejorar el rendimiento de los mismos.

2.4 Configuración de algoritmos

2.4.1 Random Forest

Podríamos hacer un estudio inicial de parámetros similar al del primer dataset, en el que retocamos “Split Criterion” y la profundidad máxima de los árboles generados. No obstante, en el conjunto de datos no vimos cambios significativos en cada configuración de parámetros.

No obstante, si vimos una gran mejora al intentar resolver el desbalanceo de clases mediante un submuestreo de la clase mayoritaria. Es por ello que voy a realizar lo mismo en este caso.

Volvemos a realizar un submuestreo y no un sobremuestreo ya que el nodo SMOTE ejecutado una vez por cada fold en la validación cruzada es demasiado costoso y lento.

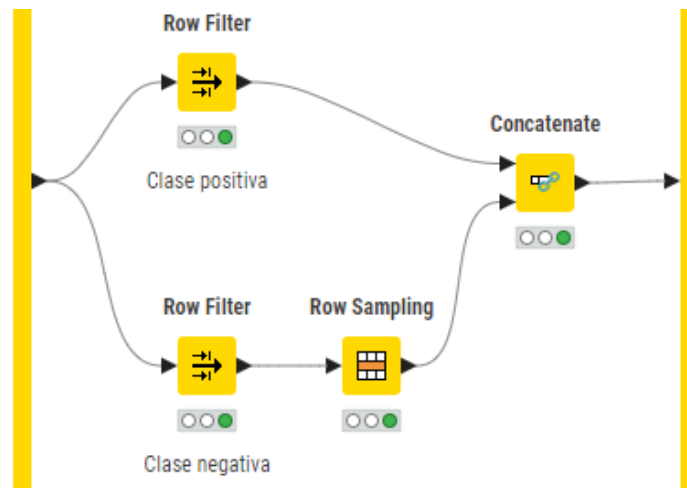


Figura 23. Flujo de datos para el submuestreo de la clase mayoritaria.

Como mostramos en la Figura 23, hemos dividido los datos en dos tablas, cada una conteniendo los registros con clase positiva y negativa, respectivamente. Hemos comprobado cuántos registros había en la clase minoritaria (unos 1400) y hemos filtrado registros en la clase mayoritaria (negativa) tal que ahora las dos clases tendrán el mismo número de registros. Finalmente, hemos unido las dos tablas de nuevo.

Realizando estos cambios podemos ahora hacer una comparativa del rendimiento realizando el submuestreo y sin realizarlo.

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,861 | 0,905 |
| AUC | 0,854 | 0,965 |

Tabla 11. Comparación del rendimiento del algoritmo Random Forest, entre aplicar o no submuestreo a la clase mayoritaria.

Alejandro Coman Venceslá

Podemos apreciar en la Tabla 11 que tanto la precisión como, sobre todo, el valor del AUC, el valor del AUC aumenta significativamente. Este enfoque será el que vayamos a aplicar en el resto de algoritmos, ya que en el apartado 2.3 de esta memoria, hemos concluido que todos los algoritmos tienen problemas al tratar con datos con clases desbalanceadas.

Mostramos los nuevos resultados tras aplicar estas optimizaciones.

| Clase / Predicción | <i>match (predicción) = 1</i> | <i>match (predicción) = 0</i> |
|--------------------|-------------------------------|-------------------------------|
| <i>match = 1</i> | 1194 | 186 |
| <i>match = 0</i> | 77 | 1303 |

Tabla 13. Matriz de confusión de Random Forest optimizado.

| match | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|---------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 1194 | 77 | 1303 | 186 | 0.865 | 0.94 | 0.865 | 0.944 | 0.901 | |
| 0 | 1303 | 186 | 1194 | 77 | 0.944 | 0.875 | 0.944 | 0.865 | 0.908 | |
| Overall | | | | | | | | | | 0,905 |

Tabla 14. Estadísticas de precisión de Random Forest optimizado.

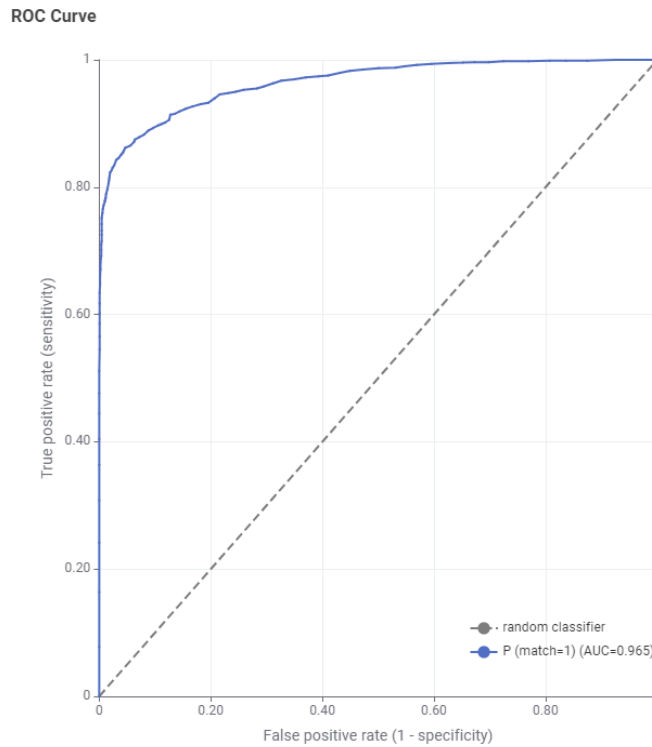


Figura 24. Nueva curva ROC para Random Forest optimizado.

2.4.2 Naive Bayes

Ya que en el estudio realizado en el primer conjunto de datos de aprobación de créditos concluimos que la configuración óptima era normalizar los datos y ya hemos aplicado la normalización en el preprocesamiento general de este conjunto de datos. Vamos a estudiar únicamente cómo mejora el rendimiento aplicando de nuevo el submuestreo de la clase mayoritaria.

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,772 | 0,775 |
| AUC | 0,859 | 0.86 |

Tabla 15. Comparación del rendimiento del algoritmo Naive Bayes, entre aplicar o no submuestreo a la clase mayoritaria

Podemos comprobar que realizar un submuestreo de la clase mayoritaria en este conjunto de datos, para el algoritmo Naive Bayes, no presenta mejoras significativas en el rendimiento de

Ya que no hemos cambiado ninguna configuración para este algoritmo, no necesitamos mostrar de nuevo los resultados, pues ya fueron mostrados en el apartado 2.3.2.

2.4.3 XGBoost

Ya que XGBoost también tiene el problema de una alta cantidad de falsos negativos, debido al desbalanceo de clases. Y viendo el éxito que ha tenido este método tanto en el primer conjunto de datos de aprobación de créditos como en el algoritmo Random Forest para este conjunto de datos, comprobaremos si también mejora el rendimiento para XGBoost.

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0.853 | 0.905 |
| AUC | 0,841 | 0.965 |

Tabla 16. Comparación del rendimiento de XGBoost, entre aplicar o no submuestreo a la clase mayoritaria.

En la Tabla 16, donde comparamos los rendimientos entre aplicar o no el submuestreo de la clase negativa de “match”, podemos ver como mejora el rendimiento al aplicar esta técnica. Vamos a ver los nuevos resultados obtenidos con más detalle:

| Clase / Predicción | <i>match (predicción) = 1</i> | <i>match (predicción) = 0</i> |
|---------------------------|-------------------------------|-------------------------------|
| <i>match = 1</i> | 1213 | 167 |
| <i>match = 0</i> | 94 | 1286 |

Tabla 17. Matriz de confusión de XGBoost optimizado.

| match | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|---------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 1213 | 94 | 1286 | 167 | 0.879 | 0.928 | 0.879 | 0.932 | 0.903 | |
| 0 | 1286 | 167 | 1213 | 94 | 0.932 | 0.885 | 0.932 | 0.879 | 0.909 | |
| Overall | | | | | | | | | | 0,905 |

Tabla 18. Estadísticas de precisión de XGBoost optimizado.

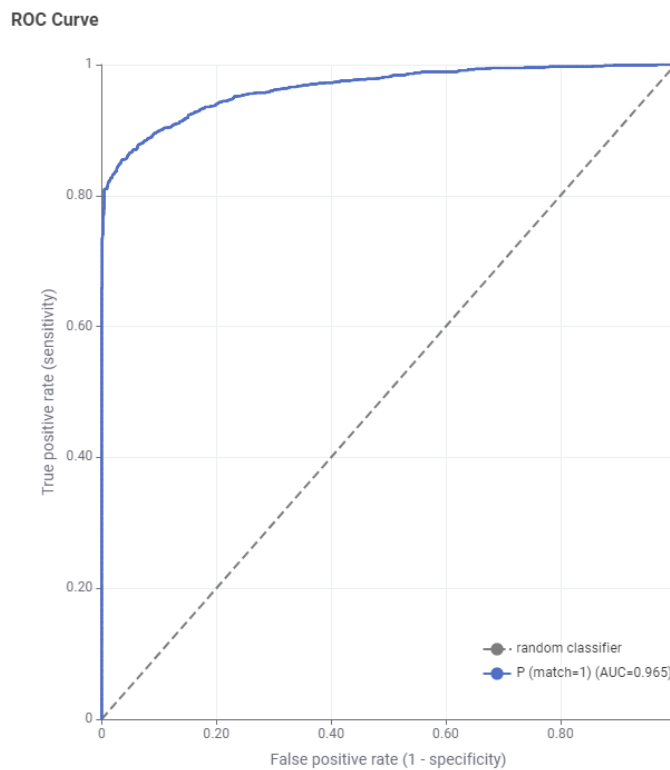


Figura 25. Nueva Curva ROC para XGBoost Optimizado.

2.4.4 Gradient Boosted Trees

Al ser XGBoost una implementación más eficiente de XGBoost y haber tenido un gran incremento en el rendimiento aplicando el submuestreo, vamos a volver a aplicarlo en este algoritmo directamente.

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,862 | 0,911 |
| AUC | 0,856 | 0,969 |

Tabla 19. Comparación de rendimiento de Gradient Boosted Trees, al aplicar submuestreo.

Alejandro Coman Venceslá

De nuevo vemos un gran incremento en el rendimiento al aplicar el submuestreo, aunque no es nada sorprendente ya que si en XGBoost lo pudimos ver, era de esperar que aquí también. Nuevos resultados con más detalle:

| Clase / Predicción | <i>match (predicción) = 1</i> | <i>match (predicción) = 0</i> |
|--------------------|-------------------------------|-------------------------------|
| <i>match = 1</i> | 1214 | 166 |
| <i>match = 0</i> | 79 | 1301 |

Tabla 20. Matriz de confusión de Gradient Boosted Trees optimizado.

| match | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|---------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 1214 | 79 | 1301 | 166 | 0.878 | 0.939 | 0.88 | 0.943 | 0.908 | |
| 0 | 1301 | 166 | 1214 | 79 | 0.943 | 0.887 | 0.943 | 0.88 | 0.914 | |
| Overall | | | | | | | | | | 0.911 |

Tabla 21. Estadísticas de precisión de Gradient Boosted Trees optimizado.

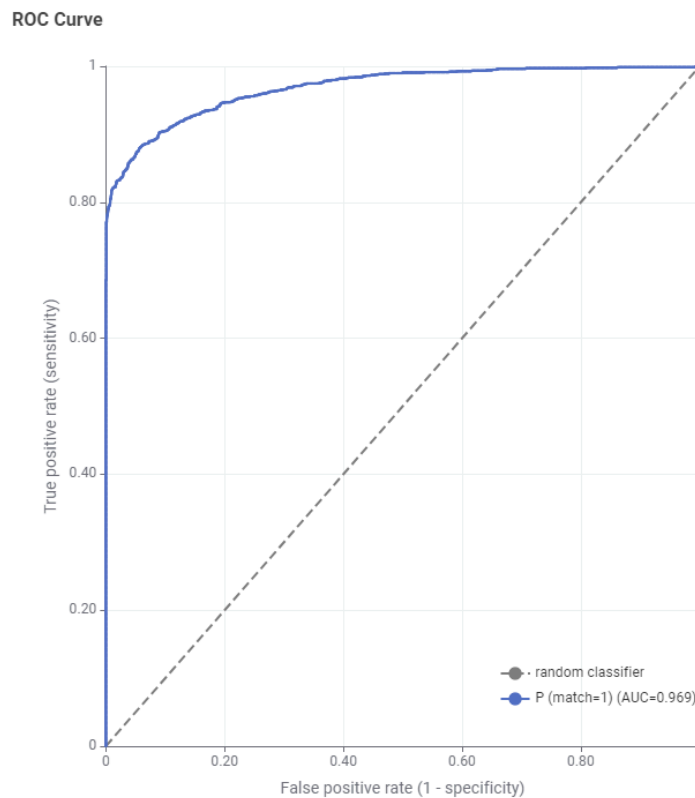


Figura 26. Nueva Curva ROC para el Gradient Boosted Trees Optimizado.

Alejandro Coman Venceslá

2.4.5 Stochastic Gradient Descent

Volvemos a estudiar si aplicando el submuestreo logramos mejorar el rendimiento

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,858 | 0,859 |
| AUC | 0,849 | 0,93 |

Tabla 22. Comparación de rendimiento de SGD, al aplicar submuestreo.

Podemos ver que el valor de AUC mejora al aplicar submuestreo, volviendo a comprobar que esta técnica para solucionar el problema del desbalanceo de clases funciona. Nuevos resultados:

| Clase / Predicción | <i>match (predicción) = 1</i> | <i>match (predicción) = 0</i> |
|--------------------|-------------------------------|-------------------------------|
| <i>match = 1</i> | 1178 | 202 |
| <i>match = 0</i> | 187 | 1193 |

Tabla 23. Matriz de confusión de SGD optimizado.

| match | TP | FP | TN | FN | Recall | Precisión | Sensitivity | Specifity | F | Acc. |
|---------|------|-----|------|-----|--------|-----------|-------------|-----------|-------|-------|
| 1 | 1178 | 187 | 1193 | 202 | 0.854 | 0.863 | 0.854 | 0.864 | 0.858 | |
| 0 | 1193 | 202 | 1178 | 187 | 0.864 | 0.855 | 0.864 | 0.854 | 0.86 | |
| Overall | | | | | | | | | | 0.859 |

Tabla 24. Estadísticas de precisión de SGD optimizado.

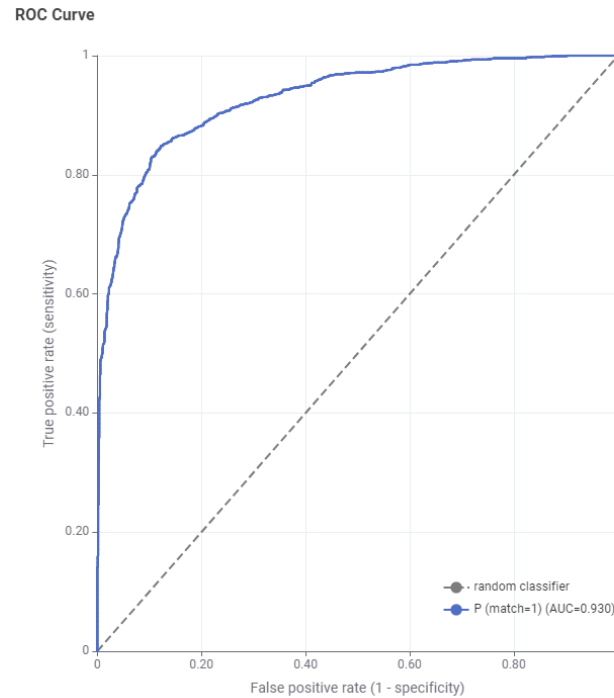


Figura 27. Nueva Curva ROC para el SGD optimizado.

Una vez hemos terminado de aplicar optimizaciones a todos los algoritmos vamos a ver en una sola gráfica como han quedado las nuevas curvas ROC, comparándolas con las antiguas.

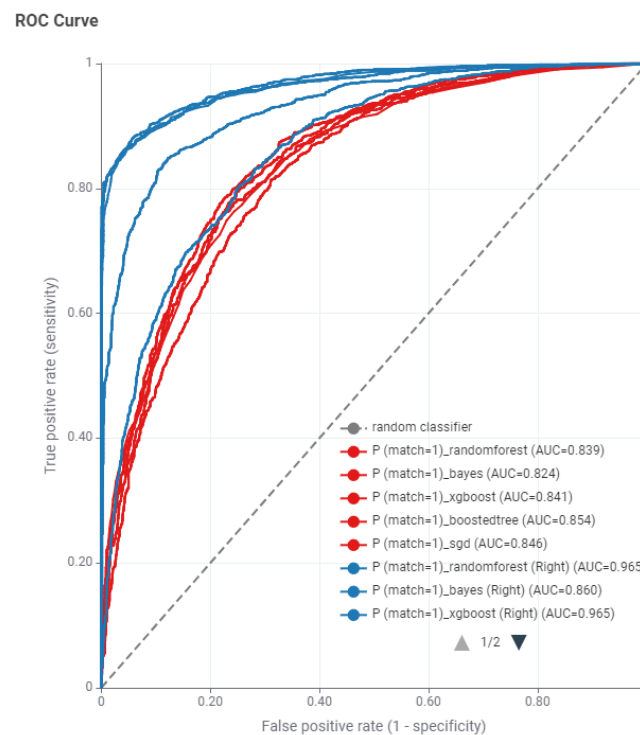


Figura 28. Comparación de curvas ROC entre los 5 algoritmos, optimizados y sin optimizar.

Alejandro Coman Venceslá

Como podemos ver en la Figura 28, donde comparamos las curvas ROC de los algoritmos sin optimizar y con optimizaciones (las curvas en azul son con optimizaciones y en rojo sin optimizaciones), nuestras mejoras han podido incrementar de manera notable el rendimiento de los algoritmos, ya que las curvas ROC nuevas se encuentran por encima de las de los algoritmos sin optimizar.

2.5 Análisis de resultados

Procedemos ahora a mostrar una tabla resumen con todos los algoritmos. Aquí incluiremos la tabla estadística del nodo Scorer y el valor AUC para cada algoritmo.

| | TP | FP | TN | FN | TPR | TNR | Accuracy | AUC |
|-------------------------------|------|-----|------|------|-------|-------|----------|-------|
| Random Forest | 1194 | 77 | 1303 | 186 | 0,865 | 0,944 | 0,905 | 0,965 |
| Naive Bayes | 5468 | 376 | 1004 | 1530 | 0,781 | 0,728 | 0,772 | 0,835 |
| XGBoost | 1213 | 94 | 1286 | 167 | 0,879 | 0,928 | 0,905 | 0,965 |
| Gradient Boosted Trees | 1214 | 79 | 1301 | 166 | 0,88 | 0,943 | 0,911 | 0,969 |
| SGD | 1178 | 187 | 1193 | 202 | 0,854 | 0,864 | 0,859 | 0,93 |

Tabla 25. Datos estadísticos de los 5 algoritmos.

Comparación entre algoritmos. TPR, TNR, Accuracy y AUC

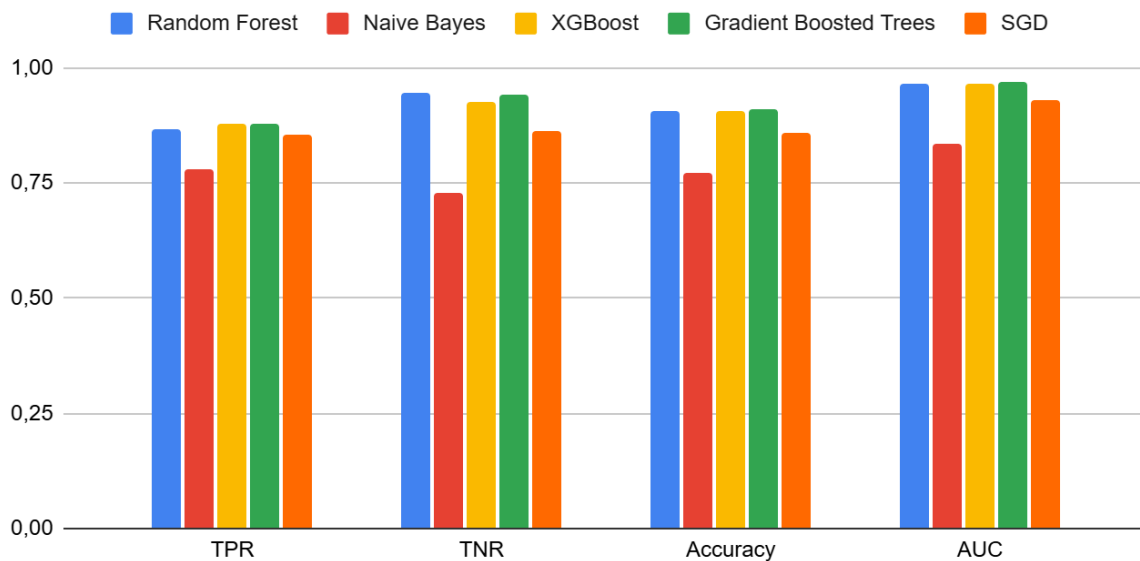


Figura 29. Gráfica comparativa de los 5 algoritmos, basada en la tabla 25.

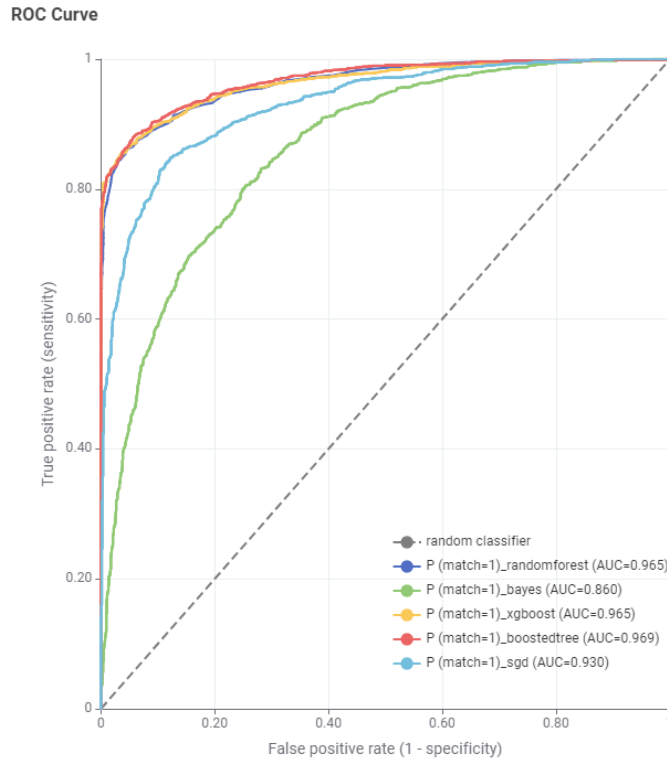


Figura 30. Curvas ROC de los cinco algoritmos.

Como podemos observar en los datos expuestos y en la gráfica y curvas ROC construidas a partir de los datos anteriores, en las Figuras 33 y 34, tenemos claro lo siguiente:

Gradient Boosted Trees, Random Forest y XGBoost destacan en este conjunto de datos debido a que son métodos de ensamble, propensos a proporcionar mejores resultados. Estos modelos de ensamble funcionan combinando varios árboles de decisión, lo que les permite capturar patrones no lineales y adaptarse a interacciones sutiles entre variables.

Finalmente, **SGD** y **Naive Bayes** muestran un rendimiento inferior debido a sus limitaciones estructurales. **SGD**, que es una técnica de optimización que se adapta bien a modelos lineales, no captura de manera adecuada las relaciones complejas que pueden existir entre una cantidad elevada de variables, además tiende a sobre ajustarse más de la cuenta. **Naive Bayes**, por su parte, asume independencia condicional entre variables, lo cual es una suposición poco realista que pocas veces se suele dar, y más con una cantidad alta de variables. En este conjunto de datos tenemos variables como por ejemplo la edad y la diferencia de edad de la pareja, las cuales tienen cierto grado de dependencia. Esto se da con más variables en el modelo.

2.6 Interpretación de los datos

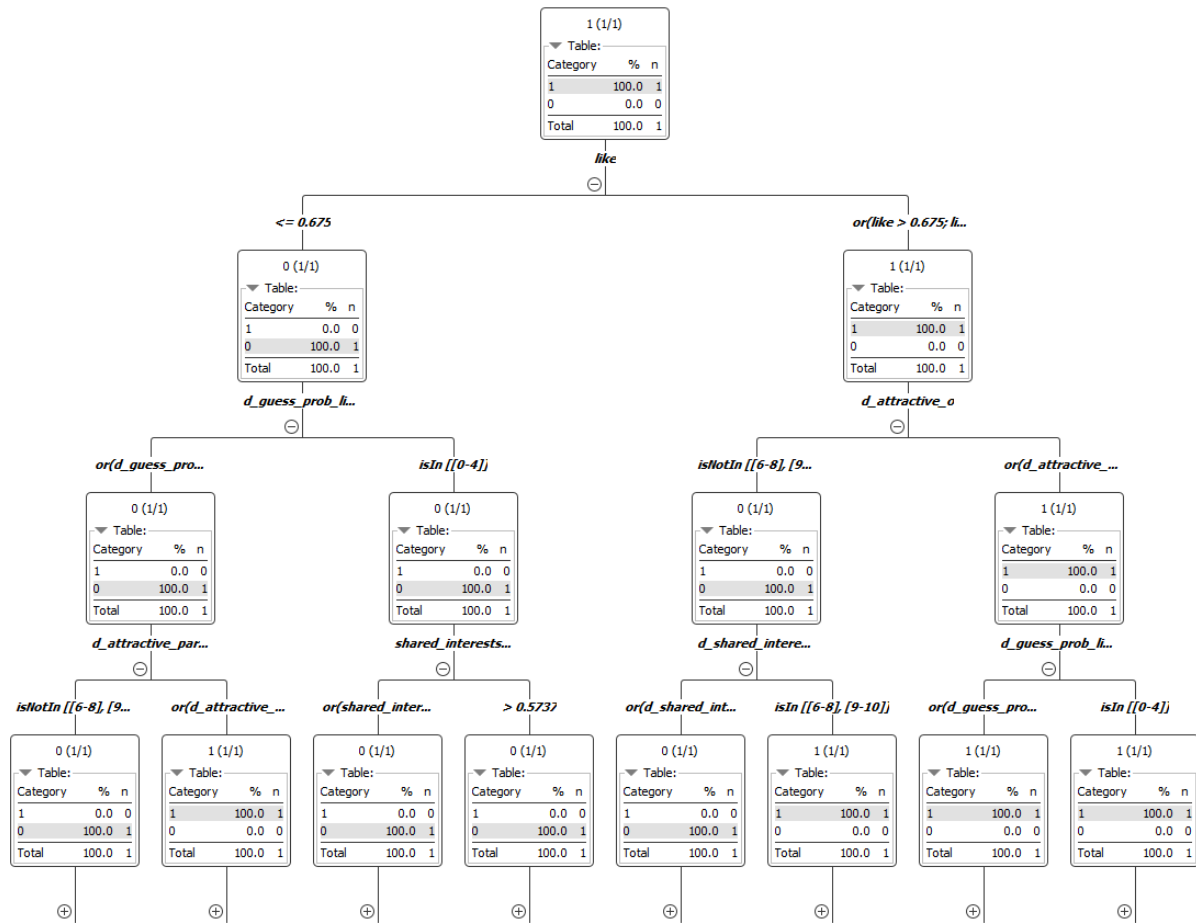


Figura 31. Árbol de decisión (modelo 1) de Random Forest

Se ha extraído en la Figura 31 uno de los árboles construidos con el algoritmo de Random Forest. Podemos ver como las variables principales elegidas para dividir el conjunto de datos han sido ("like"), la atracción percibida (variables como "d_attractive_o" y "d_attractive_partner") y los intereses compartidos ("shared_interests"). La primera división se realiza en función de "like", que actúa como un indicador inicial de afinidad; si el valor de "like" supera un umbral de 0.675, es más probable que ocurra un "match". A partir de ahí, el modelo profundiza en otras características, como la percepción de atracción mutua y los intereses en común, para refinar su predicción. En general, altos valores en gusto inicial, atracción mutua y similitud de intereses aumentan significativamente la probabilidad de "match". Este enfoque permite al modelo capturar la importancia de la percepción inicial y la compatibilidad en la decisión final de "match" entre dos personas.

Alejandro Coman Venceslá

2.7 Contenido adicional

2.8 Bibliografía

- <https://ccia.ugr.es/~casillas/knime.html>
- <https://hub.knime.com/vittoriohaardt/spaces/Public/Cross%20Validation%20example~f3DLUOsU8xclZBjc/current-state>
- https://docs.knime.com/2018-12/analytics_platform_quickstart_guide/index.html#start-knime-analytics-platform
- https://hub.knime.com/knime/spaces/Examples/02_ETL_Data_Manipulation/01_Filtering/03_Row_Filtering~zzjm_YTvol2flBdP/most-recent
- <https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.preproc.correlation.filter.CorrelationFilterNodeFactory>
- https://hub.knime.com/knime/extensions/org.knime.features.ext.weka_3.7/latest
- Prado de la Asignatura

3. Tipo de enfermedad eritemato-escamosa

3.1. Introducción

El conjunto de datos "Dermatology" tiene como objetivo predecir el tipo de enfermedad eritemato-escamosa, un problema común y complejo en dermatología. Este dataset incluye 366 instancias y 34 atributos, de los cuales 33 son valores enteros y uno es categórico. Las enfermedades en este grupo (psoriasis, dermatitis seborreica, liquen plano, pitiriasis rosada, dermatitis crónica y pitiriasis rubra pilaris) comparten síntomas clínicos como eritema y descamación, lo que dificulta su diagnóstico diferencial. Aunque el diagnóstico suele requerir una biopsia, estas enfermedades comparten características histopatológicas, complicando aún más el proceso. Cada paciente fue evaluado clínicamente en 12 características y mediante una biopsia en 22 características histopatológicas, analizadas a través de muestras microscópicas. El dataset incluye una variable para el historial familiar, que indica si alguna de estas enfermedades ha sido observada en familiares. Además, contiene valores faltantes y un posible desbalance de clases, factores que se deben considerar en la modelización para mejorar la precisión del diagnóstico.

Consideraciones acerca de este dataset:

- **No hay variables categóricas.** Este conjunto de datos está compuesto por variables numéricas. Los atributos representan características clínicas y histopatológicas con valores enteros que indican la presencia o intensidad de cada característica. No se encuentran variables categóricas en el dataset.
- **Valores faltantes solo en una clase.** De acuerdo con la información disponible, los valores faltantes se encuentran únicamente en una clase, lo cual puede tener implicaciones para el análisis y el modelado, ya que estos valores ausentes podrían influir en el rendimiento del modelo al predecir esa clase específica.
- **Desbalance de clases.** Existe un desbalance de clases en este conjunto de datos. La distribución de instancias es la siguiente:
 - **Psoriasis:** 112 instancias
 - **Dermatitis seborreica:** 61 instancias
 - **Liquen plano:** 72 instancias
 - **Pitiriasis rosada:** 49 instancias
 - **Dermatitis crónica:** 52 instancias
 - **Pitiriasis rubra pilaris:** 20 instancias

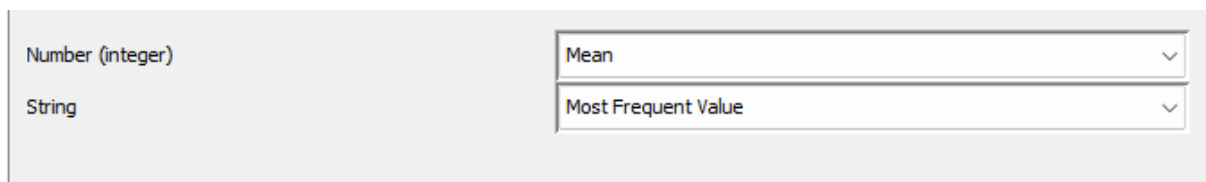
La clase más representada es **psoriasis**, mientras que la menos representada es **pitiriasis rubra pilaris**, con solo 20 instancias. Esto podría afectar al modelo de clasificación, que podría sesgar hacia las clases mayoritarias. Es recomendable aplicar técnicas para manejar el desbalance de clases, como la ponderación de clases, el submuestreo de las clases mayoritarias o el sobremuestreo de las minoritarias para mejorar la precisión del modelo en las clases minoritarias.

Para realizar la predicción de este problema he seleccionado los siguientes algoritmos:

1. **K-Nearest Neighbors:** algoritmo basado en instancias clasifica un caso en función de la similitud con sus "vecinos" más cercanos. En este caso, puede ser útil si las características clínicas e histopatológicas agrupan bien a las diferentes enfermedades, permitiendo que los casos similares se asocien con la misma clase. Dado el desbalance de clases, KNN puede sesgarse hacia la clase mayoritaria, por lo que sería necesario aplicar técnicas de balance de clases o ajuste en el número de vecinos para mejorar su rendimiento. También es sensible a la escala de las características, por lo que la normalización de los datos sería importante.
2. **Gradient Boosted Trees:** potente algoritmo de ensamblado basado en boosting, donde los árboles se construyen secuencialmente, corrigiendo los errores del árbol anterior. Este enfoque suele ser muy efectivo para capturar relaciones complejas en los datos y es menos propenso al sobreajuste en comparación con un solo árbol.
3. **Random Forest:** algoritmo de ensamblado que utiliza múltiples árboles de decisión entrenados con diferentes subconjuntos de los datos, mejorando la precisión y la capacidad de generalización. Es especialmente útil en problemas donde puede haber ruido en las características y puede capturar bien relaciones no lineales.
4. **Árbol de Decisión:** Los árboles de decisión son fáciles de interpretar y permiten visualizar las reglas que el modelo utiliza para tomar decisiones. Esto es particularmente útil en problemas de clasificación médica, donde la interpretabilidad es importante para comprender qué características son clave para distinguir entre las distintas enfermedades.
5. **Support Vector Machine (SVM):** algoritmo potente para la clasificación multiclase, especialmente cuando las clases son difíciles de separar. Utiliza un margen máximo para separar las clases y es eficaz en problemas de alta dimensionalidad. Además, puede trabajar con características no lineales mediante el uso de kernels.

3.2. Procesado de datos

En este conjunto de datos no tenemos la presencia de valores extremos ya que la mayoría de datos son valores comprendidos entre 0 y 1. Ya que solamente tenemos algunos valores faltantes en la edad, vamos a simplemente rellenarlos con un "Missing Values".



| | |
|------------------|---------------------|
| Number (integer) | Mean |
| String | Most Frequent Value |

Figura 1. Configuración del nodo Missing Value para el preprocesado de los datos.

Una vez tratado con los valores faltantes, he realizado un estudio de correlación lineal del modelo y simplemente he eliminado variables con un "Correlation Filter" con un límite de 0.7:

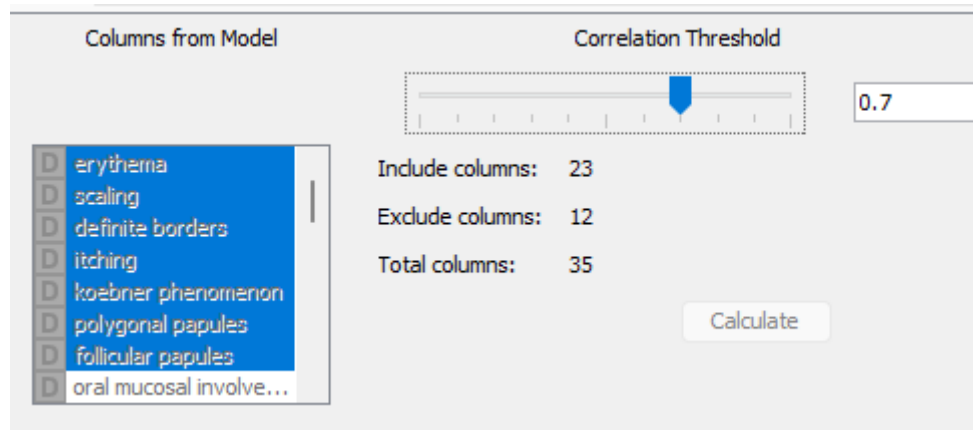


Figura 2. Filtrado de correlación lineal en el preprocesado general de los datos. Nodo "Correlation Filter"

En la Figura 2 podemos observar que se han filtrado unas 12 variables fuera, quedándonos con 23 variables (contando con la variable clase, por lo que tenemos 22 variables dependientes).

Finalmente he normalizado todos los datos, aunque ya se encontraban comprendidos entre 0 y 1. No obstante he decidido normalizarlos ya que he creído conveniente normalizar también la edad, la cual tiene un rango mucho más grande que el resto de datos. Creo que es importante que las variables tengan rangos similares de valores. El nodo de preprocesamiento general para este conjunto de datos queda de la siguiente manera:

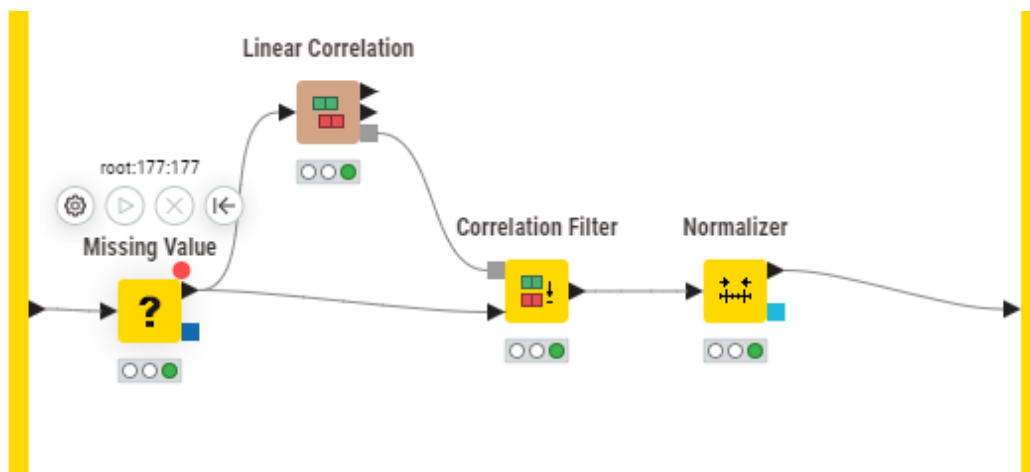


Figura 3. Flujo de datos para el preprocesamiento general.

Alejandro Coman Venceslá

El resto del flujo de datos se estructura de la siguiente manera:

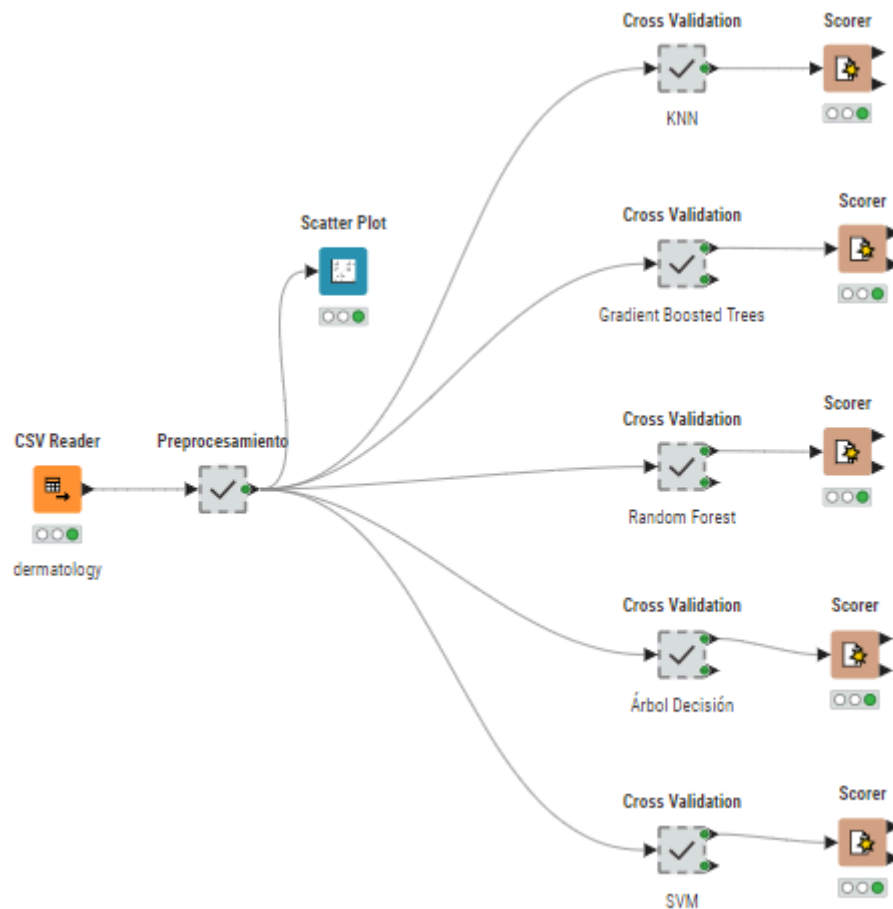


Figura 4. Vista general del flujo de trabajo para el problema de dermatología.

Tras leer los datos del archivo CSV y aplicar el preprocesamiento anteriormente mencionado, mandaremos estos datos a cada algoritmo, el cual tendrá una validación cruzada con 5 validaciones. Todos los nodos de validación cruzada (X-Partitioner y X-Aggregator) tienen la siguiente configuración:

| | |
|-----------------------|----------------------------------|
| Number of validations | 5 |
| Linear sampling | <input type="radio"/> |
| Random sampling | <input type="radio"/> |
| Stratified sampling | <input checked="" type="radio"/> |
| Class column | class |

| | |
|-------------------|--------------------|
| Target column | class |
| Prediction column | Prediction (class) |

Figuras 5 y 6. Configuración de los nodos X-Partitioner (izquierda) y X-Aggregator (derecha) para la validación cruzada.

Alejandro Coman Venceslá

En las Figuras 5 y 6 muestro la configuración de los nodos encargados de la validación cruzada. Se ha utilizado “stratified sampling” basado en la clase objetivo, para intentar aminorar el problema del desbalanceo de las clases.

3.3 Resultados obtenidos

A continuación vamos a comprobar los resultados que hemos obtenido para cada uno de los algoritmos, habiendo aplicado este preprocesamiento general y habiendo dejado la configuración de los algoritmos por defecto. Mostraremos las matrices de confusión y las tablas de estadísticas de los nodos Scorer. También podremos analizar las primeras métricas que obtenemos de cada algoritmo.

3.3.1 KNN

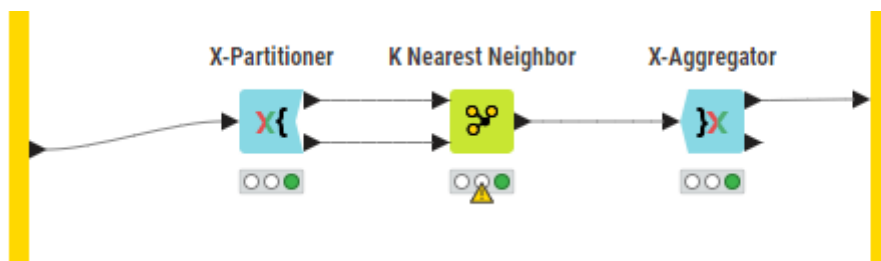


Figura 7. Metanodo de la validación cruzada para el algoritmo “KNN”.

| RowID | 2 Number (integer) | 1 Number (integer) | 3 Number (integer) | 5 Number (integer) | 4 Number (integer) | 6 Number (integer) |
|-------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 2 | 53 | 0 | 0 | 0 | 8 | 0 |
| 1 | 1 | 111 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 70 | 0 | 2 | 0 |
| 5 | 0 | 0 | 0 | 52 | 0 | 0 |
| 4 | 1 | 0 | 1 | 0 | 47 | 0 |
| 6 | 0 | 0 | 0 | 2 | 3 | 15 |

Tabla 1. Matriz de confusión de KNN.

| RowID | TruePositives Number (integer) | FalsePositives Number (integer) | TrueNegatives Number (integer) | FalseNegatives Number (integer) | Recall Number (double) | Precision Number (double) | Sensitivity Number (double) | Specificity Number (double) | F-measure Number (double) | Accuracy Number (double) |
|---------|-----------------------------------|------------------------------------|-----------------------------------|------------------------------------|---------------------------|------------------------------|--------------------------------|--------------------------------|------------------------------|-----------------------------|
| 2 | 53 | 2 | 303 | 8 | 0.869 | 0.964 | 0.869 | 0.993 | 0.914 | ② |
| 1 | 111 | 0 | 254 | 1 | 0.991 | 1 | 0.991 | 1 | 0.996 | ② |
| 3 | 70 | 1 | 293 | 2 | 0.972 | 0.986 | 0.972 | 0.997 | 0.979 | ② |
| 5 | 52 | 2 | 312 | 0 | 1 | 0.963 | 1 | 0.994 | 0.981 | ② |
| 4 | 47 | 13 | 304 | 2 | 0.959 | 0.783 | 0.959 | 0.959 | 0.862 | ② |
| 6 | 15 | 0 | 346 | 5 | 0.75 | 1 | 0.75 | 1 | 0.857 | ② |
| Overall | ② | ② | ② | ② | ② | ② | ② | ② | ② | 0.951 |

Tabla 2. Estadísticas de precisión de KNN.

Al tratarse de un problema de clasificación multiclase, nos es más complicado representar el rendimiento en una curva ROC, no obstante podemos fijarnos que en la matriz de confusión obtenemos unas clasificaciones muy buenas, ya que en la mayoría de clases clasificamos correctamente los ejemplos. Además tenemos una precisión de 0,951.

3.3.2 Gradient Boosted Trees

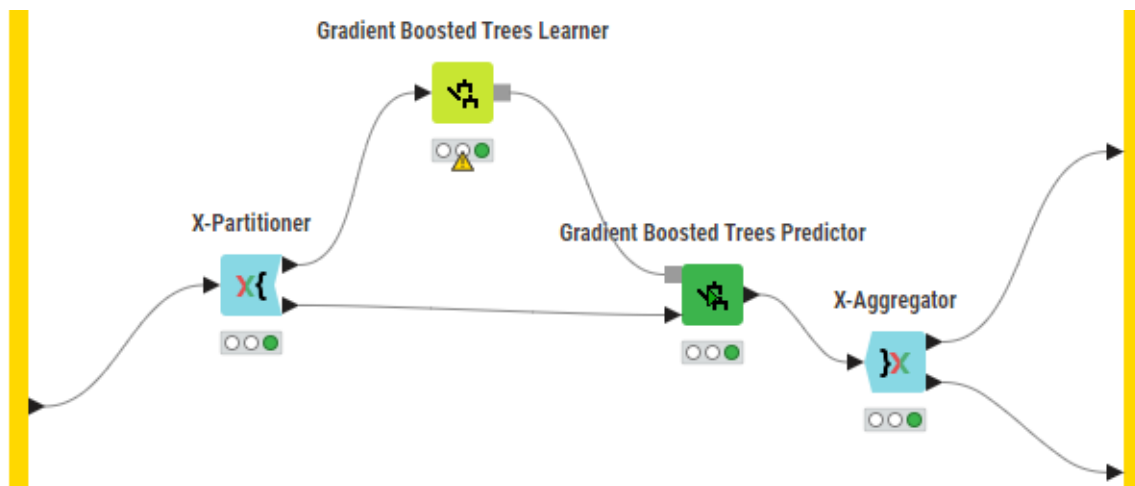


Figura 8. Metanodo de la validación cruzada para el algoritmo “Gradient Boosted Trees”.

| RowID | 2 Number (integer) | 1 Number (integer) | 3 Number (integer) | 5 Number (integer) | 4 Number (integer) | 6 Number (integer) |
|-------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 2 | 51 | 2 | 0 | 0 | 8 | 0 |
| 1 | 1 | 111 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 69 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 52 | 0 | 0 |
| 4 | 6 | 1 | 0 | 0 | 42 | 0 |
| 6 | 0 | 2 | 0 | 0 | 1 | 17 |

Tabla 3. Matriz de confusión de Gradient Boosted Trees.

| RowID | TruePositives Number (integer) | FalsePositives Number (integer) | TrueNegatives Number (integer) | FalseNegatives Number (integer) | Recall Number (double) | Precision Number (double) | Sensitivity Number (double) | Specificity Number (double) | F-measure Number (double) | Accuracy Number (double) |
|---------|-----------------------------------|------------------------------------|-----------------------------------|------------------------------------|---------------------------|------------------------------|--------------------------------|--------------------------------|------------------------------|-----------------------------|
| 2 | 51 | 8 | 297 | 10 | 0.836 | 0.864 | 0.836 | 0.974 | 0.85 | ② |
| 1 | 111 | 6 | 248 | 1 | 0.991 | 0.949 | 0.991 | 0.976 | 0.969 | ② |
| 3 | 69 | 0 | 294 | 3 | 0.958 | 1 | 0.958 | 1 | 0.979 | ② |
| 5 | 52 | 0 | 314 | 0 | 1 | 1 | 1 | 1 | 1 | ② |
| 4 | 42 | 10 | 307 | 7 | 0.857 | 0.808 | 0.857 | 0.968 | 0.832 | ② |
| 6 | 17 | 0 | 346 | 3 | 0.85 | 1 | 0.85 | 1 | 0.919 | ② |
| Overall | ② | ② | ② | ② | ② | ② | ② | ② | ② | 0.934 |

Tabla 4. Estadísticas de precisión de Gradient Boosted Trees.

En este escenario, tras ver las Tablas 3 y 4, podemos observar como el algoritmo Gradient Boosted Trees tiene Accuracy algo más bajo (0.934), el cual sigue siendo muy bueno. Podemos ver como las instancias clasificadas correctamente no difieren mucho del algoritmo anterior. Podemos ver quizá que la clase 4 ha tenido algunas instancias más que han sido clasificadas incorrectamente (falsos clase 4). Intentaremos ver más adelante como podemos mejorar el rendimiento.

3.3.3 Random Forest

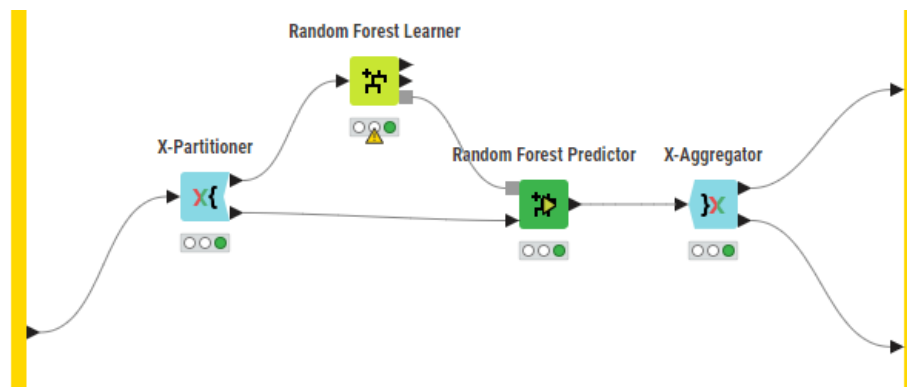


Figura 9. Metanodo de la validación cruzada para Random Forest.

| RowID | 2 Number (integer) | 1 Number (integer) | 3 Number (integer) | 5 Number (integer) | 4 Number (integer) | 6 Number (integer) |
|-------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 2 | 56 | 0 | 0 | 0 | 4 | 1 |
| 1 | 0 | 112 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 70 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 52 | 0 | 0 |
| 4 | 5 | 1 | 0 | 0 | 43 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 | 18 |

Tabla 5. Matriz de confusión de Random Forest.

| RowID | TruePositives Number (integer) | FalsePositives Number (integer) | TrueNegatives Number (integer) | FalseNegatives Number (integer) | Recall Number (double) | Precision Number (double) | Sensitivity Number (double) | Specificity Number (double) | F-measure Number (double) | Accuracy Number (double) |
|---------|-----------------------------------|------------------------------------|-----------------------------------|------------------------------------|---------------------------|------------------------------|--------------------------------|--------------------------------|------------------------------|-----------------------------|
| 2 | 56 | 7 | 298 | 5 | 0.918 | 0.889 | 0.918 | 0.977 | 0.903 | ② |
| 1 | 112 | 2 | 252 | 0 | 1 | 0.982 | 1 | 0.992 | 0.991 | ② |
| 3 | 70 | 0 | 294 | 2 | 0.972 | 1 | 0.972 | 1 | 0.986 | ② |
| 5 | 52 | 0 | 314 | 0 | 1 | 1 | 1 | 1 | 1 | ② |
| 4 | 43 | 5 | 312 | 6 | 0.878 | 0.896 | 0.878 | 0.984 | 0.887 | ② |
| 6 | 18 | 1 | 345 | 2 | 0.9 | 0.947 | 0.9 | 0.997 | 0.923 | ② |
| Overall | ② | ② | ② | ② | ② | ② | ② | ② | ② | 0.959 |

Tabla 6. Estadísticas de precisión de Random Forest.

En el algoritmo de Random Forest, como podemos ver en las Tablas 5 y 6, hemos tenido una precisión de 0.959, de momento la mejor de todas. No obstante, fijarse solamente en la métrica de precisión no es una buena práctica. Si nos fijamos también en la matriz de confusión, podemos ver que ahora tenemos menos clases del tipo 2 erróneamente clasificadas como la clase 4. Esto puede ser un buen indicador de que el clasificador es algo mejor.

Alejandro Coman Venceslá

3.3.4 Árbol Decisión

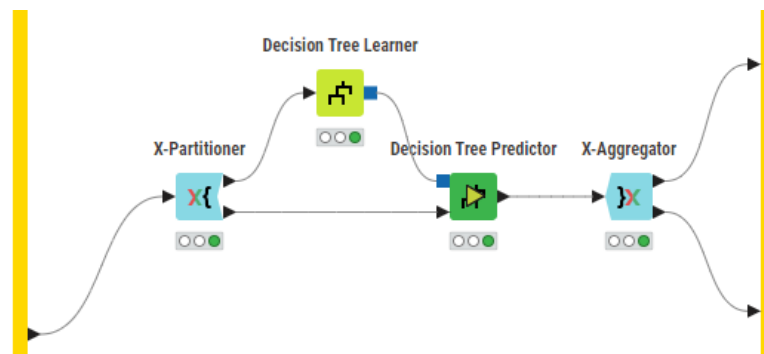


Figura 10. Metanodo de la validación cruzada para el Árbol de Decisión.

| RowID | 2 Number (integer) | 1 Number (integer) | 3 Number (integer) | 5 Number (integer) | 4 Number (integer) | 6 Number (integer) |
|-------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 2 | 57 | 2 | 0 | 0 | 1 | 1 |
| 1 | 3 | 108 | 0 | 0 | 0 | 1 |
| 3 | 1 | 1 | 67 | 1 | 2 | 0 |
| 5 | 0 | 0 | 0 | 52 | 0 | 0 |
| 4 | 5 | 1 | 0 | 0 | 43 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 | 18 |

Tabla 7. Matriz de confusión de Árbol de Decisión.

| RowID | TruePositives Number (integer) | FalsePositives Number (integer) | TrueNegatives Number (integer) | FalseNegativ... Number (integer) | Recall Number (double) | Precision Number (double) | Sensitivity Number (double) | Specificity Number (double) | F-measure Number (double) | Accuracy Number (double) |
|---------|-----------------------------------|------------------------------------|-----------------------------------|-------------------------------------|---------------------------|------------------------------|--------------------------------|--------------------------------|------------------------------|-----------------------------|
| 2 | 57 | 10 | 295 | 4 | 0.934 | 0.851 | 0.934 | 0.967 | 0.891 | ② |
| 1 | 108 | 5 | 249 | 4 | 0.964 | 0.956 | 0.964 | 0.98 | 0.96 | ② |
| 3 | 67 | 0 | 294 | 5 | 0.931 | 1 | 0.931 | 1 | 0.964 | ② |
| 5 | 52 | 1 | 313 | 0 | 1 | 0.981 | 1 | 0.997 | 0.99 | ② |
| 4 | 43 | 3 | 314 | 6 | 0.878 | 0.935 | 0.878 | 0.991 | 0.905 | ② |
| 6 | 18 | 2 | 344 | 2 | 0.9 | 0.9 | 0.9 | 0.994 | 0.9 | ② |
| Overall | ② | ② | ② | ② | ② | ② | ② | ② | ② | 0.943 |

Tabla 8. Estadísticas de precisión de Árbol de Decisión.

En este escenario, en las Tablas 7 y 8, podemos ver como la métrica de precisión también es alta, con un valor de 0.943. Además, fijándonos en la matriz de confusión, podemos ver que casi todas las instancias han sido también clasificadas correctamente.

3.3.5 Support Vector Machine

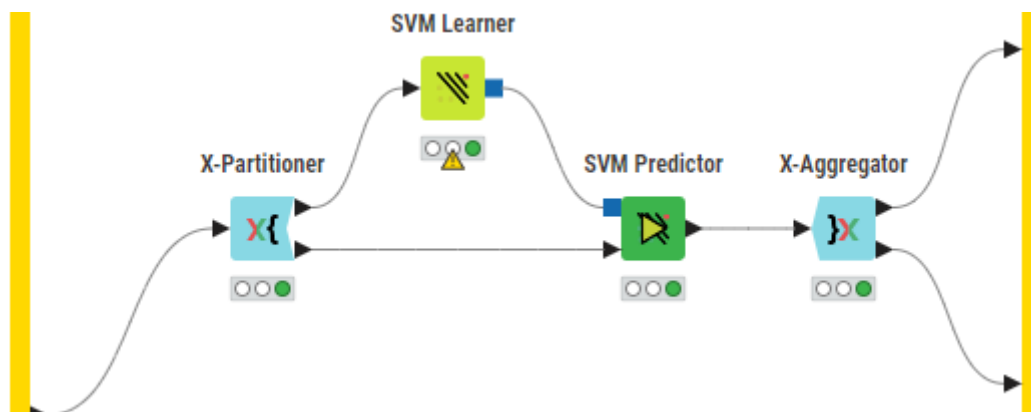


Figura 11. Metanodo de la validación cruzada para el algoritmo “SVM”.

| RowID | 2 Number (integer) | 1 Number (integer) | 3 Number (integer) | 5 Number (integer) | 4 Number (integer) | 6 Number (integer) |
|-------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 2 | 58 | 0 | 0 | 0 | 3 | 0 |
| 1 | 0 | 112 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 69 | 0 | 3 | 0 |
| 5 | 0 | 0 | 0 | 51 | 0 | 1 |
| 4 | 3 | 0 | 1 | 0 | 45 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 20 |

Tabla 9. Matriz de confusión de SVM.

| RowID | TruePositives Number (integer) | FalsePositives Number (integer) | TrueNegatives Number (integer) | FalseNegatives Number (integer) | Recall Number (double) | Precision Number (double) | Sensitivity Number (double) | Specificity Number (double) | F-measure Number (double) | Accuracy Number (double) |
|---------|-----------------------------------|------------------------------------|-----------------------------------|------------------------------------|---------------------------|------------------------------|--------------------------------|--------------------------------|------------------------------|-----------------------------|
| 2 | 58 | 3 | 302 | 3 | 0.951 | 0.951 | 0.951 | 0.99 | 0.951 | ② |
| 1 | 112 | 0 | 254 | 0 | 1 | 1 | 1 | 1 | 1 | ② |
| 3 | 69 | 1 | 293 | 3 | 0.958 | 0.986 | 0.958 | 0.997 | 0.972 | ② |
| 5 | 51 | 0 | 314 | 1 | 0.981 | 1 | 0.981 | 1 | 0.99 | ② |
| 4 | 45 | 6 | 311 | 4 | 0.918 | 0.882 | 0.918 | 0.981 | 0.9 | ② |
| 6 | 20 | 1 | 345 | 0 | 1 | 0.952 | 1 | 0.997 | 0.976 | ② |
| Overall | ② | ② | ② | ② | ② | ② | ② | ② | ② | 0.97 |

Tabla 10. Estadísticas de precisión de SVM.

Como se puede observar en las Tablas 9 y 10. Tenemos la precisión más alta de nuevo, con un valor de 0.97. Además también tenemos casi todas las instancias clasificadas correctamente en la matriz de confusión. Podemos afirmar que tenemos un buen clasificador también.

Figura 18. Curva ROC de Gradient Boosted Trees.

Como podemos observar en las Tablas 7 y 8 y en la Figura 18, hemos obtenido en este escenario una precisión con valor 0.862 y un área debajo de la curva (AUC) con valor 0.856, las cuales son algo mejores que XGBoost. Debemos recordar que XGBoost es una implementación supuestamente optimizada de Gradient Boosted Trees. No obstante, estamos probando nodos con sus valores por defecto de momento. Más adelante cuando hagamos algunos retoques a los modelos intentaremos mejorar los rendimientos de los clasificadores.

3.4 Configuración de algoritmos

3.4.1 KNN

Comprobaremos si el rendimiento del algoritmo mejora aplicando un submuestreo de las clases mayoritarias, ya que también tenemos un problema de clases desbalanceadas en este conjunto de datos.

Hemos realizado el siguiente submuestreo de clases:

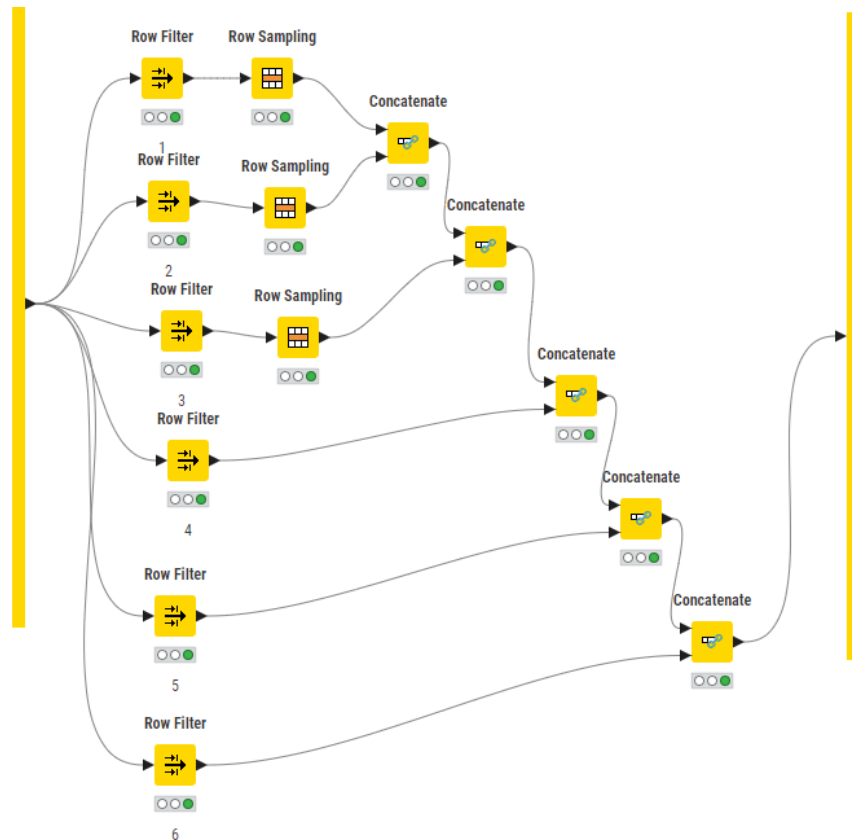


Figura 12. Submuestreo de clases mayoritarias.

En la Figura 12 aplicamos el submuestreo de clases mayoritarias. Hemos limitado el número de instancias de cada clase a 50. Como las clases con número significativamente superior a 50 son solamente las clases 1, 2 y 3, hemos submuestreado solamente estas, dejando intactas las otras tres.

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,951 | 0,926 |

Tabla 11. Comparación del rendimiento del algoritmo KNN, entre aplicar o no submuestreo a la clases mayoritarias

Ya que la precisión no mejora tras aplicar submuestreo, como mostramos en la Tabla 11. No lo aplicaremos en este algoritmo

3.4.2 Gradient Boosted Trees

Ya que en el estudio realizado en el primer y segundo conjunto de datos el submuestreo fue altamente eficaz, vamos a volver a aplicarlo

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,934 | 0,934 |

Tabla 12. Comparación del rendimiento del algoritmo Gradient Boosted Trees, entre aplicar o no submuestreo a la clases mayoritarias

Podemos comprobar que realizar un submuestreo de la clase mayoritaria en este conjunto de datos, para el algoritmo Gradient Boosted Trees no mejora la precisión, es por ello que lo dejaremos en la configuración por defecto. No obstante, ya presenta buenos resultados de por sí.

3.4.3 Random Forest

Aplicando el submuestreo, vemos si hay alguna diferencia en el rendimiento en el algoritmo Random Forest

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0.959 | 0.956 |

Tabla 13. Comparación del rendimiento de Random Forest, entre aplicar o no submuestreo a la clase mayoritaria.

Podemos ver que el rendimiento no mejora, sino que incluso empeora un poco. Es por ello que nos quedamos con la configuración por defecto. No obstante una precisión de 0.959 es ya de por sí casi perfecta.

3.4.4 Árbol Decisión

Aplicando el submuestreo, vemos si hay alguna diferencia en el rendimiento en el Árbol de Decisión.

| | Sin submuestreo | Con submuestreo |
|------------------|-----------------|-----------------|
| Precisión | 0,943 | 0,93 |

Tabla 14. Comparación de rendimiento del Árbol de Decisión, al aplicar submuestreo.

De nuevo, nos damos cuenta de que el rendimiento no mejora, y también empeora un poco. Es por ello que nos quedamos con la configuración por defecto. No obstante, una precisión de 0.93 es muy buena.

Alejandro Coman Venceslá

3.4.5 Support Vector Machine

Volvemos a estudiar si aplicando el submuestreo logramos mejorar el rendimiento

| | Sin submuestreo | Con submuestreo |
|-----------|-----------------|-----------------|
| Precisión | 0,97 | 0,959 |

Tabla 15. Comparación de rendimiento de SGD, al aplicar submuestreo.

De nuevo, observamos que el rendimiento no mejora, y también empeora un poco. Es por ello que nos quedamos con la configuración por defecto.

3.5 Análisis de resultados

Procedemos ahora a mostrar una tabla resumen con todos los algoritmos, mostrando la precisión de cada algoritmo

| | KNN | Gradient Boosted Trees | Random Forest | Árbol Decisión | SVM |
|-----------|-------|------------------------|---------------|----------------|------|
| Precisión | 0.951 | 0.934 | 0.959 | 0.943 | 0.97 |

Tabla 25. Comparación de la precisión de los 5 algoritmos.

A partir de los resultados de precisión obtenidos para los cinco algoritmos en esta tarea de clasificación multiclase, podemos concluir que el algoritmo **SVM** obtuvo la precisión más alta, con un valor de **0.97**, seguido de **Random Forest** con **0.959**. En el contexto de una clasificación multiclase en el diagnóstico diferencial de enfermedades eritemato-escamosas, un mayor valor de precisión es deseable, ya que indica una mayor proporción de predicciones correctas en relación con el total de predicciones realizadas. Esto es especialmente importante en el ámbito médico, donde un error de clasificación puede llevar a diagnósticos incorrectos y, por tanto, a tratamientos inadecuados.

En comparación con los otros algoritmos, un modelo con una precisión más alta es preferible porque reduce la probabilidad de clasificaciones incorrectas, lo cual es crucial en un problema de clasificación multiclase y aún más cuando se trata de diagnosticar enfermedades. En este caso, **SVM** es el algoritmo que muestra el mejor desempeño y sería una elección adecuada para esta tarea, siempre que los recursos computacionales lo permitan, ya que tiende a ser más intensivo en procesamiento para conjuntos de datos grandes.

En adición, los otros algoritmos (Gradient Boosted Trees, KNN, y Árbol de Decisión) presentan limitaciones que reducen su precisión: GBT es sensible a los parámetros y al ruido, KNN tiene problemas con la escala y la superposición de clases, y el Árbol de Decisión tiende a sobreajustarse.

3.6 Interpretación de los datos

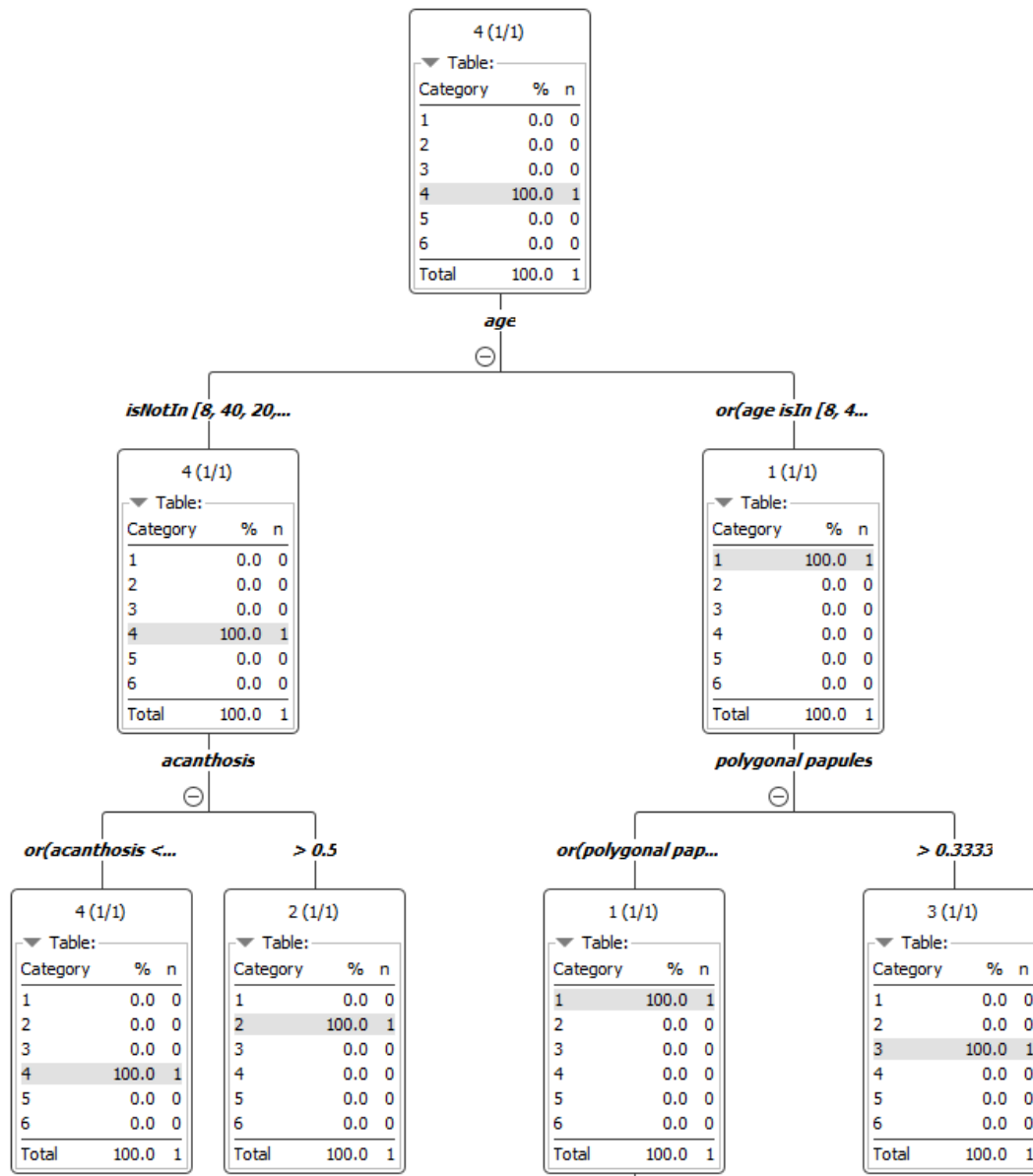


Figura 13. Árbol de decisión (modelo 1) de Random Forest

Como podemos observar en la Figura 13, que nos muestra unos de los modelos generados por el algoritmo Random Forest, destacan la “**edad**” y las características clínicas “**acanthosis**” (engrosamiento de la epidermis) y “**polygonal papules**” como factores clave en el diagnóstico diferencial de enfermedades eritemato-escamosas. La edad se utiliza como el primer criterio de división, sugiriendo que ciertos rangos etarios se asocian con tipos específicos de enfermedades. Posteriormente, características como el nivel de acanthosis y la presencia de papulones poligonales ayudan a refinar la predicción en función de su presencia o intensidad. En conjunto, estos factores permiten al modelo realizar una clasificación preliminar efectiva que es luego afinada por los demás árboles en el bosque, mejorando la precisión del diagnóstico.

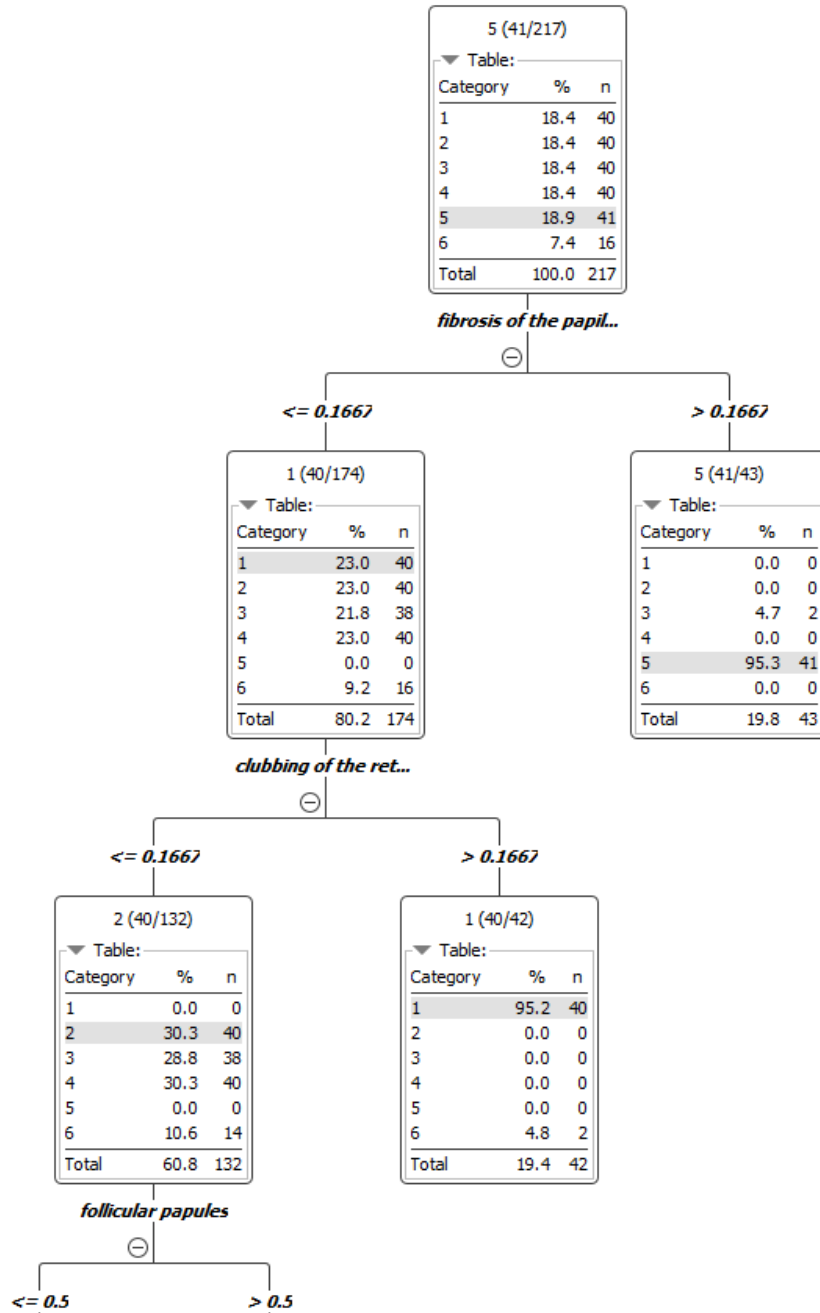


Figura 14. Modelo generado por el Árbol de Decisión

Este árbol de decisión del modelo Random Forest muestra que la variable “**fibrosis of the papillae**” es el primer criterio de división, indicando su importancia en la clasificación de las enfermedades. Si el valor de fibrosis de las papilas es menor o igual a 0.1667, el árbol evalúa la variable “**clubbing of the rete ridges**”, una característica que permite distinguir aún más entre las clases. En la rama izquierda, una siguiente división se realiza según “**follicular papules**”, refinando la predicción en función de su presencia. En la rama derecha, para fibrosis de las papilas superior a 0.1667, el modelo clasifica principalmente en una categoría específica. Este árbol sugiere que la fibrosis de las papilas, el clubbing de las crestas reticulares y las pápulas foliculares son características clave para diferenciar entre las enfermedades eritemato-escamosas en este conjunto de datos.

3.7 Contenido adicional

3.8 Bibliografía

- <https://ccia.ugr.es/~casillas/knime.html>
- <https://hub.knime.com/vittoriohaardt/spaces/Public/Cross%20Validation%20example~f3DLUOsU8xclZBjc/current-state>
- https://docs.knime.com/2018-12/analytics_platform_quickstart_guide/index.html#start-knime-analytics-platform
- https://hub.knime.com/knime/spaces/Examples/02_ETL_Data_Manipulation/01_Filtering/03_Row_Filtering~zzjm_YTvol2flBdP/most-recent
- <https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.preproc.correlation.filter.CorrelationFilterNodeFactory>
- https://hub.knime.com/knime/extensions/org.knime.features.ext.weka_3.7/latest
- <https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.mine.knn.KnnNodeFactory2>
- https://hub.knime.com/knime/spaces/Educators%20Alliance/Guide%20to%20Intelligent%20Data%20Science/Example%20Workflows/Chapter9/03_SVM~DTfbNITUngKQVF8v/most-recent
- Prado de la Asignatura