**UNIVERSIDAD DE GRANADA**


UNIVERSIDAD DE GRANADA

Carlos de Alonso Andrés, Grupo 1
Doble Grado en Ingeniería Informática y Matemáticas
e.carlosdealonso@go.ugr.es

# Inteligencia de Negocio

Práctica 1:
Análisis Predictivo Mediante Clasificación

Curso 2022-2023

# Índice

# 1. Introducción

En esta práctica se abordarán tres problemas de clasificación del mundo real y, mediante algoritmos de clasificación supervisada vistos en clase de teoría y herramientas enseñadas en clases de prácticas, se realizará una predicción sobre ellos y se analizará como de buena es dicha clasificación. Se utilizará una validación cruzada con cinco conjuntos, todos ellos con la misma semilla, 01092000, para poder realizar la comparación adecuadamente. Toda la experimentación se realizará sobre el sistema operativo Windows y con un procesador Intel(R) Core (TM) i5-10300H CPU @ 2.50GHz 2.50 GHz.

## Heart_2020

El primer problema ha sido obtenido de la plataforma de *Kaggle* y consiste en una encuesta que realiza CDC (*Centers for Disease Control and Prevention*) cada año, con datos que corresponden a 2020 y a un total de 400.000 adultos. Aunque el *dataset* descargable contiene solamente 50.706 instancias y 18 variables como se puede ver en la Figura 1.

Se trabajará sobre datos numéricos como el IMC (Índice de Masa Corporal), PhysicalHealth, MentalHealth; y sobre datos categóricos como el sujeto fuma, bebe, hace deporte, ha tenido algún ictus… y donde la variable principal sobre la que se va a trabajar es *HeartDisease* que presenta solo dos clases “Yes” y “No” en función de si el sujeto ha padecido o no un infarto.

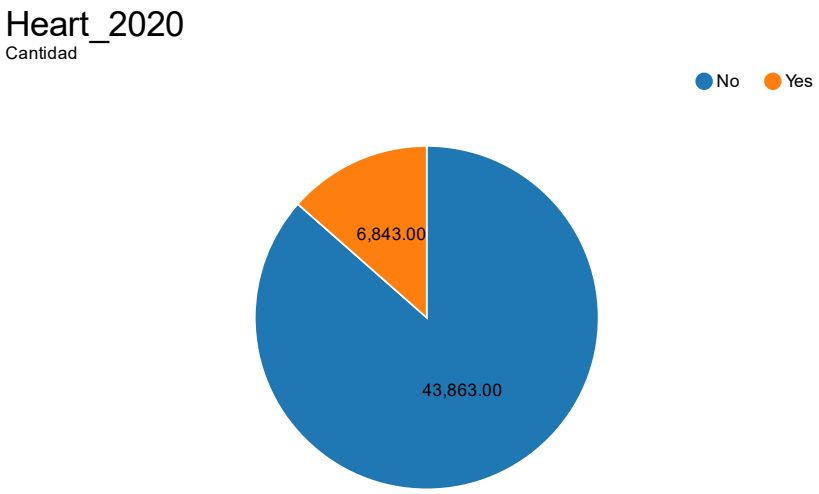


*Figura 1: Numero de instancias de HeartDisease*

El principal inconveniente que se encuentra en este problema es que la clase principal no está balanceada pues claramente se ve que un 87% de instancias corresponden a la clase “No” y un 13% a la clase “Yes”, Figura 2. Esto se resolverá en el segundo apartado. Se considerará como clase positiva “Yes”, ya que queremos predecir cuándo un sujeto padecerá un ataque al corazón.

## BodyPerformance

El segundo problema ha sido obtenido también de *Kaggle* y consiste en un problema de clasificación multiclase procesado a partir de los datos suministrados por *Korean Sports Promotion Foundation*.

En este problema todos los datos son numéricos menos el género del sujeto, algunas variables son la edad, la estatura, el peso, el porcentaje de grasa o la presión de la sangre en sístole y

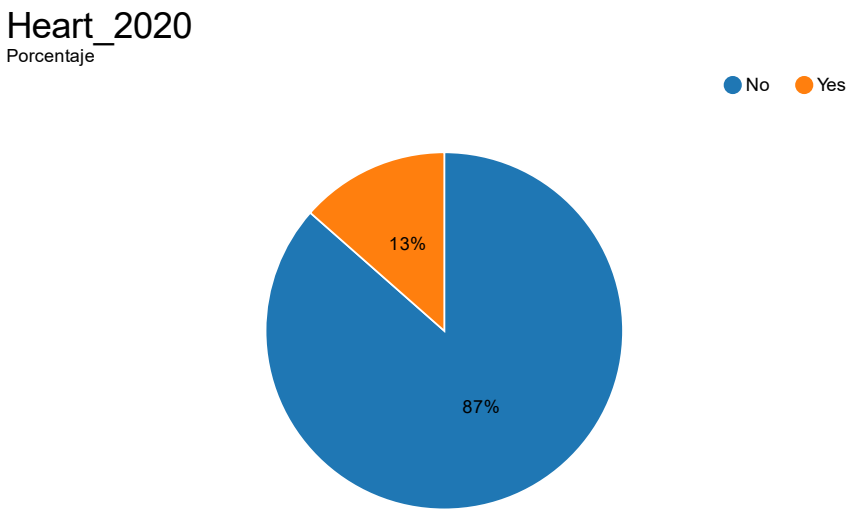


*Figura 2: Porcentaje de instancias de la variable objetivo*

y diástole… Por supuesto, al ser un problema de multiclase, la variable principal "class" puede tomar uno de los cuatro valores: "A", "B", "C" y "D"; que clasifican de mejor a peor el cuerpo del sujeto.

El problema consiste en 13393 instancias con 12 variables, algunas ya nombradas. Se puede observar en la Figura 3 que las clases están balanceadas, es decir, todas tienen el mismo número de instancias.

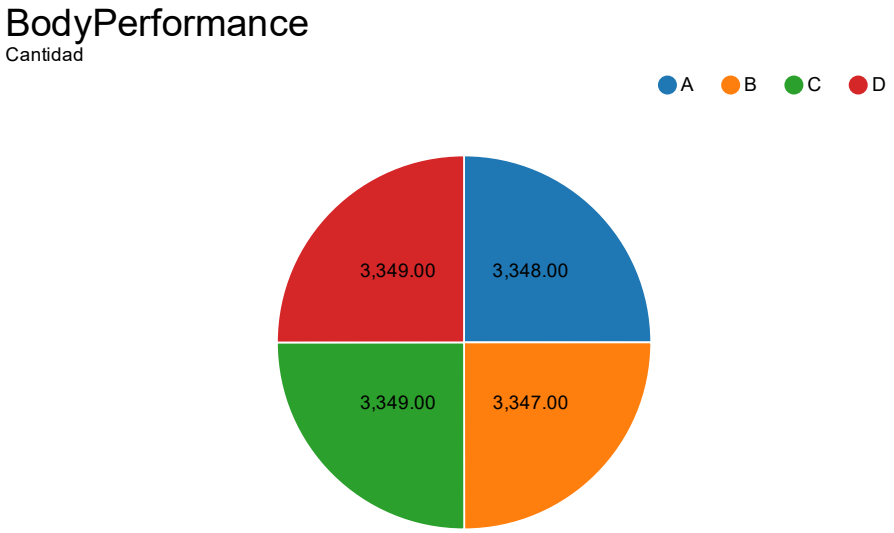


*Figura 3: Número de instancias de Class*

## Students

Por último, tenemos el problema de *students* obtenido del repositorio de *machine learning* de la UCI (*University of California, Irvine*). Su creación fue apoyada por el programa SATDAP (Sistema de Apoyo de Transformación Digital de Administración Pública) de Portugal.

Este *dataset* también sería un problema de multiclase, pues la variable a estudiar es "*Target*" y presenta tres clases distintas: si el sujeto se gradúa "*Graduate*", si el sujeto abandona "*Dropout*"

o si el sujeto está matriculado "*Enrolled*". Pero no se tendrá en cuenta esta última clase y se trabajará como una variable binaria haciendo las transformaciones que sean necesarias en el apartado de procesado de datos.

El problema consta de 4424 instancias y 37 variables, algunas de ellas son la ocupación de padre y madre, el género, el edad de matriculación, la carrera, la cantidad de créditos cursados y matriculados… Son todo datos numéricos menos las variable objetivo que es categórica.

Se puede observar que las clases no están balanceadas, como muestra la Figura 4. Además, otro inconveniente del problema es que hay una gran cantidad de valores perdidos en todas las clases, ambas dificultades serán solucionadas en el siguiente apartado.

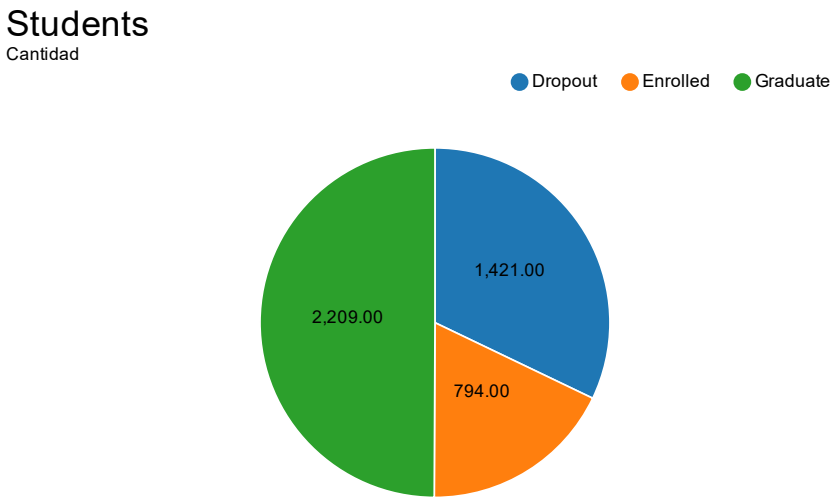


*Figura 4: Número de instancias de Target*

## 2. Procesado de Datos

Para realizar un buen estudio de los datos es necesario que todos los algoritmos partan del mismo conjunto y no presenten anomalías. Por eso, en cada problema se ha realizado una fase de procesamiento donde se ha solucionado problemas como el desbalanceo de clases, valores perdidos, eliminar clases que no se van a estudiar…

En cada problema se ha seguido un procesamiento distinto, en el primer problema se ha realizado un *oversampling* con la herramienta *SMOTE* que proporciona Knime y es una forma de solucionar el desbalanceo de clases, pero un problema que ocurre al crear tantas instancias es que se crean duplicados, luego también es necesario quitarlos para que no se realice un sobreajuste sobre esos duplicados. En la Figura 5 se puede observar un *workflow* de KNIME con un lector de archivos CSV, para extraer la información del *dataset* descargado y dos nodos mas que realizan las acciones mencionadas anteriormente.

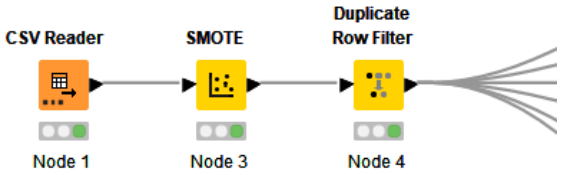


*Figura 5: Procesamiento datos Heart*

Se puede observar como el problema pasa de tener unas 40.000 instancias a ser 87.726, pero en el proceso de eliminado de duplicados desaparecen unas 5.0000, quedando el conjunto como representa la figura Figura 6.

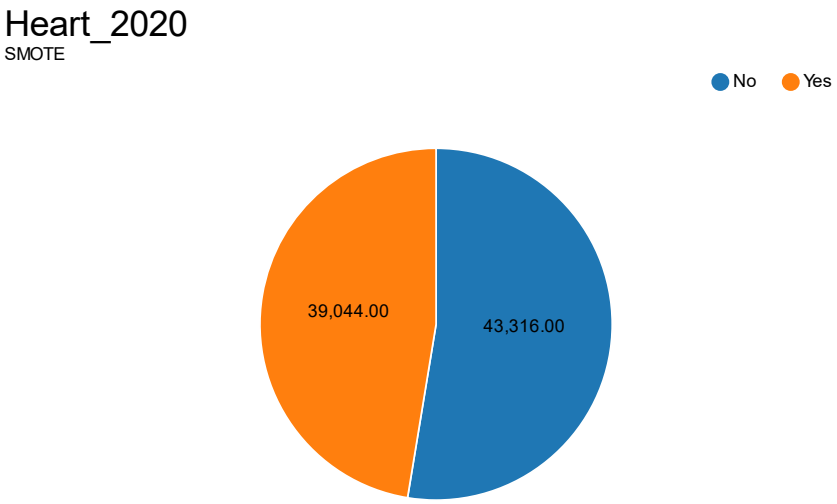


*Figura 6: Número de instancias clase HeartDisease post-procesado*

Por otro lado, en el segundo problema, ya que no hay ni desbalanceo ni valores nulos, solo se ha realizado un cambio en la variable género. Se ha transformado esta variable de una categórica a una discreta para poder utilizarla en algoritmos como redes neuronales y K-NN, ya que esta transformación no cambia el valor de la clase en la instancia sino que la traduce a otro valor que facilita los cálculos. Por ello el género masculino "M" corresponde al 0.0 y el género femenino "F" corresponde al 1.0, posteriormente la columna inicial ha sido eliminada, Figura 7.

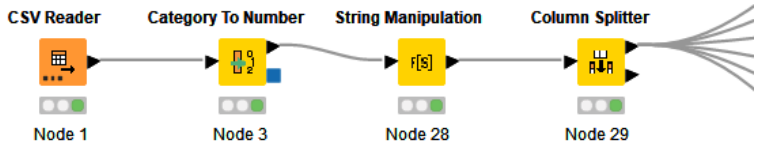


*Figura 7: Procesamiento datos BodyPerformance*

Por último, el problema de *Students*, el procesado es algo más complejo pues se tiene el problema de valores perdidos y también tenemos una clase "Enrolled", dentro de la variable objetivo que no nos interesa estudiar, luego queremos eliminarla de nuestro dominio Figura 8

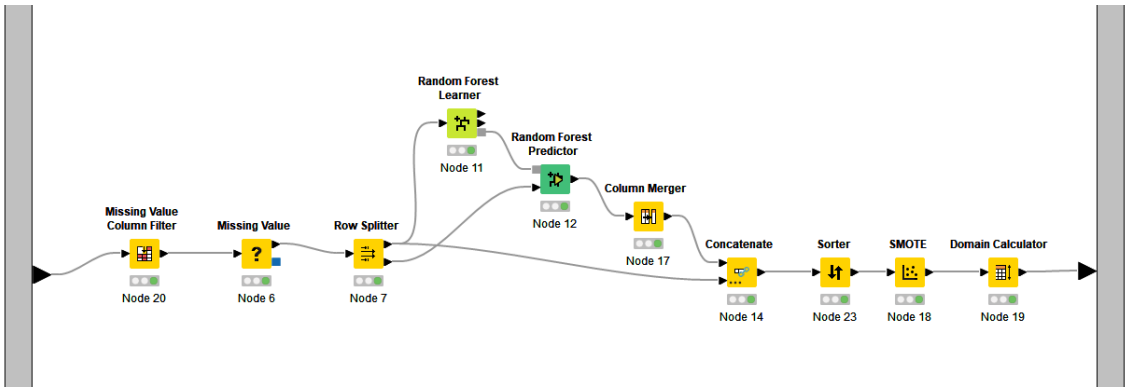


*Figura 8: Procesamiento datos Students*

Primero de todo, observamos si en nuestro *dataset* existe alguna variable que tenga un gran porcentaje de valores perdidos, como umbral se ha seleccionado que un tercio de las instancias sean nulas. Por lo que aquellas variables que cumplan la condición serán eliminadas, en este caso no existen tantos valores perdidos en ninguna de las columnas. A continuación, insertamos valores donde haga falta gracias al nodo *Missing Value*, que añadirá la mediana en los variables numéricas, este problema no tiene variables nominales luego no hace falta buscar una solución para ellas.

El siguiente paso consiste en predecir aquellas instancias que presentan la clase "Enrolled", para ello el *test data* serán las tuplas correspondientes a esta clase y el *training data* las demás, que se corresponden a las clases "Graduate" y "Dropout". Se utilizará el algoritmo de Random Forest con Índice Gini. Finalmente volveremos a juntar todas las instancias y como las clases estaban desbalanceadas será necesario realizar un *oversampling*, con el nodo SMOTE y calcular de nuevo el dominio del problema, pues se ha eliminado una clase de la variable objetivo. Luego se puede ver en Figura 9, que las clases están perfectamente balanceadas y el problema pasa de 4.424 instancias a 5.296.

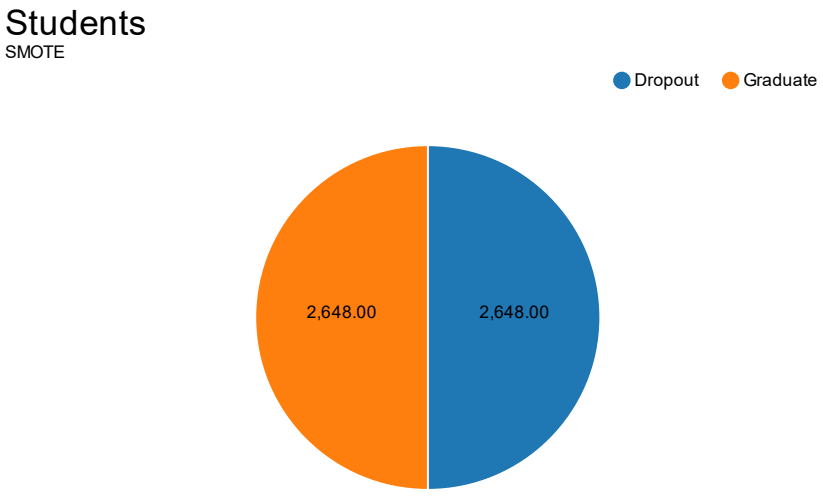


*Figura 9: Número de instancias Students post-procesado*

Con respecto al tratamiento de los datos realizados en el último problema, es posible que no sea el más adecuado para todos los algoritmos, ya se verá en el siguiente apartado. Pero otra solución podría haber sido eliminar todas esas instancias que no pertenecen a las clases que se quieren estudiar, ya que el proceso anterior puede conllevar que se arrastren errores que

perjudiquen el modelo para el mundo real, aunque en el programa se esté acertando "correctamente" las clases.

Otra cuestión que surge es, ¿qué hacer con las variables que estaban asociadas a la clase "Enrolled"? Mi decisión es mantenerlas para así, aunque se haya eliminado dicha clase, su relevancia en el modelo sigue estando. Además, tiene sentido que se estudie el comportamiento de los sujetos que están matriculados, para saber su correlación con el objeto de estudio, terminar o no la carrera.

# 3. Resultados obtenidos

## ZeroR

El proceso de clasificación mediante algoritmos comenzará siempre con este, sin ser contado en los cinco necesarios, para poder tener una cota inferior y que todos los demás algoritmos lo mejoren. Si en algún momento un clasificador obtiene resultados peores se sospecharía que se está haciendo algo mal. En la Figura 10 se tienen dos nodos que indican la validación cruzada y dos nodos más con el *learner* y el *predictor* que la extensión Weka nos facilita para utilizar ZeroR.

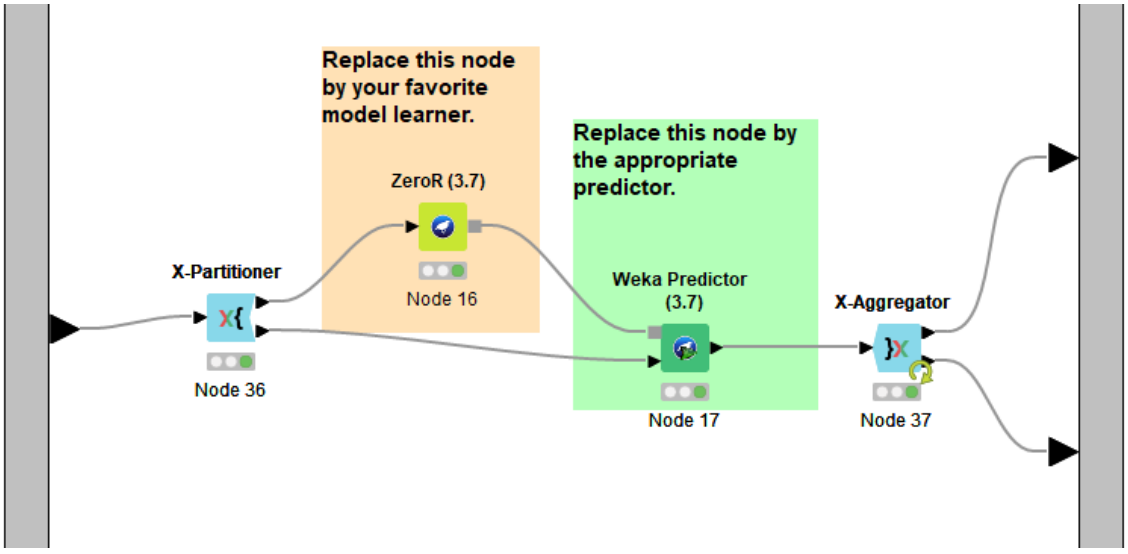


*Figura 10: ZeroR Workflow*

Este clasificador predice que cualquier instancia pertenece a la clase mayoritaria. En problemas donde el balanceo esté perfecto, los resultados obtenidos estarán entorno al 50%. Sin embargo, en el primer problema al eliminar instancias repetidas, el algoritmo obtiene una tasa de error más baja pero en ningún momento predice correctamente la clase positiva "Yes". Esto ocurre porque acierta más veces porque hay más instancias de la clase "No", pero a la vez no tiene en cuenta en ningún momento la clase "Yes". Por lo que al utilizar el nodo *Scorer* y ver su matriz de confusión, Tabla 1, todas las instancias han sido clasificadas como "No".

*Tabla 1: Matrices de confusión*

| | "No" | "Yes" |
|---|---|---|
| "No" | 43316 | 0 |
| "Yes" | 39044 | 0 |

| | "Dropout" | "Graduate" |
|---|---|---|
| "Dropout" | 1588 | 1060 |
| "Graduate" | 1590 | 1058 |

| | "C" | "A" | "B" | "D" |
|---|---|---|---|---|
| "C" | 2009 | 670 | 670 | 0 |
| "A" | 2010 | 669 | 669 | 0 |
| "B" | 2010 | 670 | 669 | 0 |
| "D" | 2007 | 670 | 670 | 0 |

En la Tabla 2, se encuentran las medidas de precisión obtenidas por el algoritmo en cada problema. Por lo mencionado anteriormente, como la clase positiva escogida en el primero ha sido "Yes", los valores de verdaderos positivos y falsos positivos serán 0. Es interesante ver que en PPV y F1-score, tienen valores son desconocidos, esto ocurre porque el número predicciones positivas correctas entre el número total de predicciones positivas es una indeterminación. PPV es la división de TP entre TP+FP y ambos valores son 0, además, como F1-score es la media armónica de PPV y TPR tampoco se puede calcular.

*Tabla 2: Medidas de precisión ZeroR*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Heart | 0 | 0 | 43316 | 39044 | 0.0 | 1.0 | ? | 0.525935 | ? | 1.0 | 0.499981 |
| Body | 669 | 2010 | 8035 | 2679 | 0.199821 | 0.7999 | 0.24972 | 0.649892 | 0.222001 | 0.999861 | |
| Stud. | 1058 | 1060 | 1588 | 1590 | 0.399547 | 0.599698 | 0.499528 | 0.499622 | 0.443978 | 0.999622 | 0.499547 |

En los demás algoritmos se han obtenido valores acorde a un buen balanceo, aunque cabe mencionar que en el problema *Body* no tenemos valor para AUC ya que al ser un problema multiclase las curvas ROC no se podrían hacer con clases positivas y negativas, habría que utilizar alguna técnica de *One vs One*.

## Naive Bayes

Este algoritmo será intercambiado por una red neuronal en los problemas con valores numéricos. Está basado en métodos bayesianos y asume que los atributos son independientes y calcula la clase más probable condicionando el resto de los atributos. Trabaja tanto con variables numéricas como con variables categóricas, pero como en este problema abundan más las segundas se ha decidido utilizar en este problema, ya que en los demás la naturaleza es distinta y tienen gran cantidad de variables continuas.

En la Figura 11, observamos que el algoritmo no necesita de un preprocesamiento antes de comenzar a modelar. Se utilizan dos nodos para la validación cruzada y dos nodos *Naive Bayes Learner* y *Naive Bayes Predictor* que son proporcionados por KNIME en la configuración por defecto.

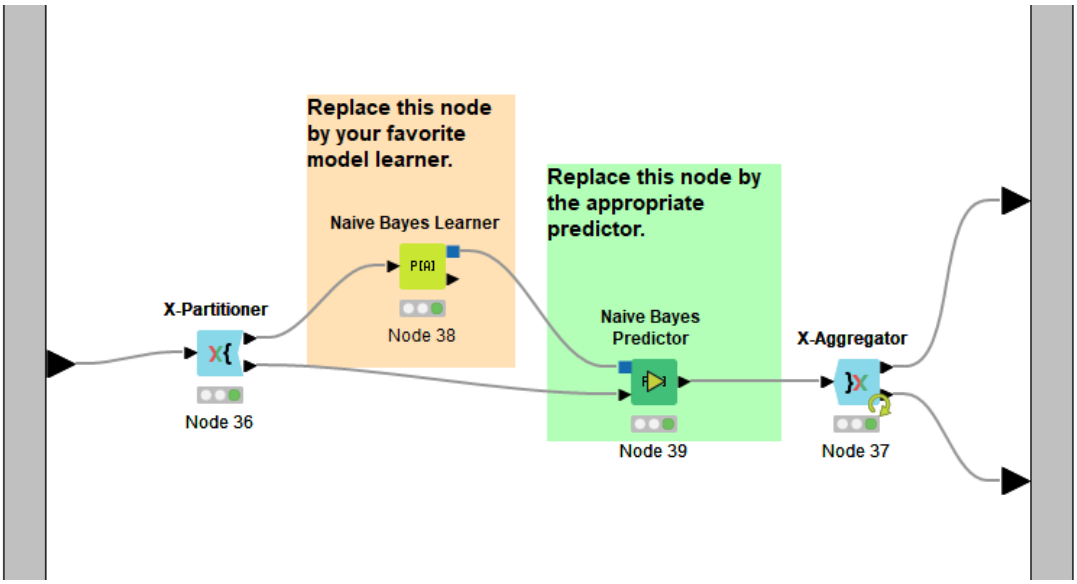


*Figura 11: Naive Bayes Workflow*

Ya ninguna presenta valores desconocidos y cómo podemos ver en la matriz de confusión, modelo intenta predecir las instancias para la clase "Yes".

*Tabla 3: Matriz de confusión Naive Bayes*

| | "No" | "Yes" |
|---|---|---|
| "No" | 14861 | 8455 |
| "Yes" | 13979 | 25065 |

En la Tabla 4: Medidas de precisión Naive Bayes, obtenemos todas las medidas de precisión que serán estudiadas.

*Tabla 4: Medidas de precisión Naive Bayes*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Heart | 25065 | 8455 | 34861 | 13979 | 0.64197 | 0.80481 | 0.74776 | 0.72761 | 0.69084 | 1.20282 | 0.8175 |

## Neural Networking

En los problemas de *Body* y de *Students*, se utiliazra una red neuronal en vez del algoritmo de Naive Bayes. En la Figura 12 observamos que los datos necesitan ser normalizados, para que unos valores no tengan más peso sobre otros, para ello usamos el nodo *Normalizer* y a continuación, los nodos para la validación cruzada y el nodo *RProp MLP Learner* y el nodo *MultiLayerPerception Predictor* que se encargaran de modelar la red neuronal, con valores por defecto.

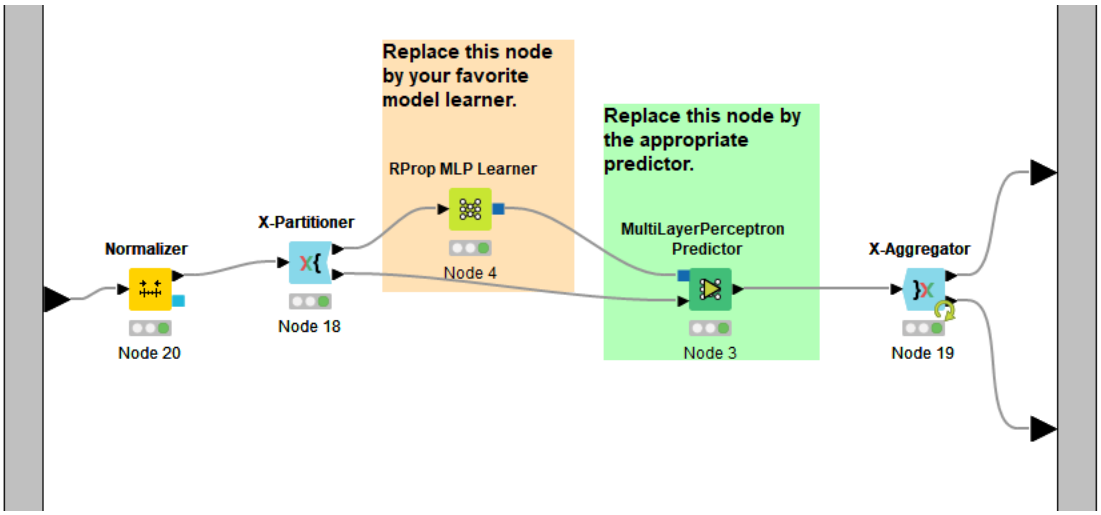


*Figura 12: Neural Network Workflow*

Las matrices de confusión de ambos problemas aparecen en la Tabla 5

*Tabla 5: Matrices de confusión Neural Network*

| | "Dropout" | "Graduate" |
|---|---|---|
| "Dropout" | 2348 | 300 |
| "Graduate" | 231 | 2417 |

| | "C" | "A" | "B" | "D" |
|---|---|---|---|---|
| "C" | 1717 | 498 | 500 | 634 |
| "A" | 105 | 2795 | 444 | 4 |
| "B" | 991 | 1293 | 931 | 132 |
| "D" | 556 | 77 | 149 | 2567 |

Las medidas de precisión que se van a estudiar son las que aparecen en la Tabla 6.

*Tabla 6: Medidas de precisión Neural Network*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Body | 2795 | 1868 | 8177 | 553 | 0.834827 | 0.814037 | 0.5994 | 0.819234 | 0.697791 | 0.697791 | |
| Stud. | 2417 | 300 | 2348 | 231 | 0.912764 | 0.886707 | 0.889584 | 0.899736 | 0.901025 | 1.341444 | 0.945225 |

## Decision Tree

El siguiente algoritmo utilizado es el árbol de decisión. Utiliza todos los ejemplos y van seleccionando atributos para dividir el conjunto de ejemplos. Como criterio para seleccionar las variables se utiliza el índice Gini, luego es un algoritmo CART.

El índice Gini mide con qué frecuencia un elemento elegido de forma aleatoria de un conjunto sería etiquetado incorrectamente si se etiqueta aleatoriamente de acuerdo con la distribución de clases en el subconjunto.

Para ello lo ejecutaremos con dos nodos que vienen por defecto en KNIME: *Decision Tree Learner* y *Decision Tree Predictor*, junto con los nodos correspondientes a la validación cruzada. En la Figura 13, se observa que tampoco será necesario un preprocesamiento de los datos.

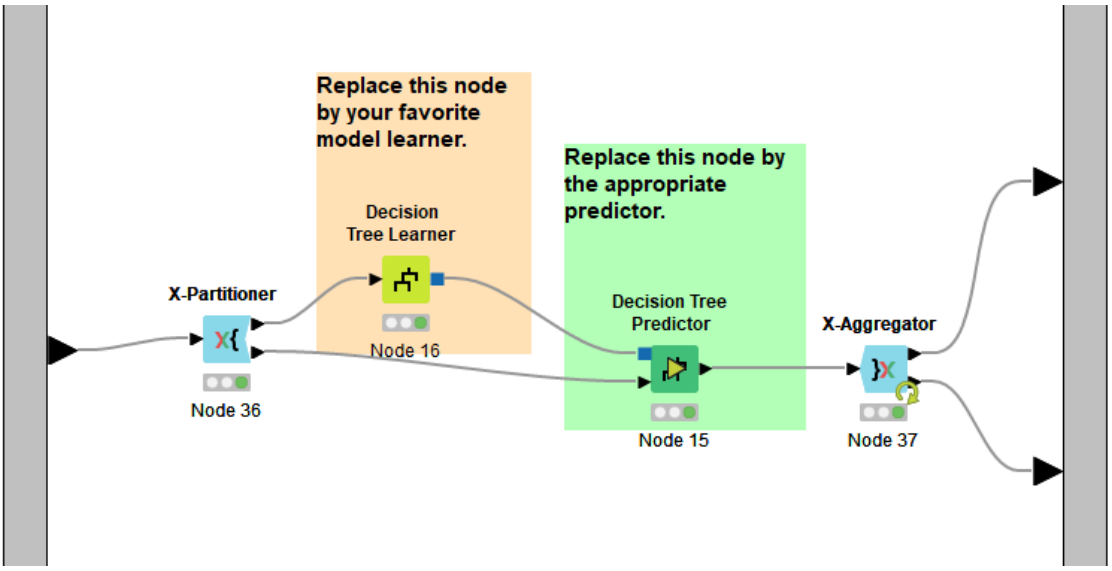


*Figura 13: Decision Tree Workflow*

En la Tabla 7 aparecen las matrices de confusión correspondientes a cada problema.

*Tabla 7: Matrices de confusión Decision Tree*

| | "No" | "Yes" |
|---|---|---|
| "No" | 36305 | 7011 |
| "Yes" | 3785 | 35259 |

| | "Dropout" | "Graduate" |
|---|---|---|
| "Dropout" | 2366 | 282 |
| "Graduate" | 297 | 2351 |

| | "C" | "A" | "B" | "D" |
|---|---|---|---|---|
| "C" | 2047 | 273 | 689 | 340 |
| "A" | 286 | 2304 | 716 | 42 |
| "B" | 695 | 628 | 1840 | 184 |
| "D" | 452 | 65 | 221 | 2611 |

En la Tabla 8 observaremos las medidas de precisión que hemos calculado.

*Tabla 8: Medidas de precisión Decision Tree*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Heart | 35259 | 7011 | 36305 | 3785 | 0.90306 | 0.83814 | 0.83414 | 0.86892 | 0.86723 | 1.31955 | 0.89105 |
| Body | 2432 | 1758 | 8287 | 916 | 0.7264 | 0.82499 | 0.58043 | 0.80034 | 0.64526 | 1.2455 | |
| Stud. | 2351 | 282 | 2366 | 287 | 0.88784 | 0.8935 | 0.8929 | 0.89067 | 0.89036 | 1.33467 | 0.89185 |

Es un algoritmo fácil de interpretar y que si un nuevo ejemplo es añadido se parte del nodo raíz hasta llegar a la clase con que se etiquetaría siguiendo la rama del árbol apropiada en cada momento. Pero por ahora no sabemos que es lo que hace con las variables continuas, se observara más adelante con una imagen del modelo del árbol.

## 5-NN

Cuando clasificamos se tiene muy en cuenta la semejanza entre ejemplo, por ello con este criterio, el algoritmo más sencillo de implementar es k-NN que realiza una predicción de una instancia en función de los k vecinos más cercanos. Se utilizará la distancia euclídea con 5 vecinos.

El preprocesamiento más completo se ha tenido que realizar en el problema *Heart* porque presenta muchas variables categóricas, pero solo se ha tenido que añadir dos nodos: *Category to Number* y *Column Splitter*, cuyo objetivo es establecer una traducción por cada clase a un valor numérico entero, de 0 hasta n-1, donde n es el número de clases que presenta la variable. Después se separan las nuevas columnas y se normalizan para un mejor funcionamiento del algoritmo.

En la Figura 14 observamos además, los nodos para la validación cruzada, y un único nodo *K Nearest Neighbor* que ya predice el conjunto *test* y por último se cambia el nombre de la variable *Prediction(#)* para que todos los algoritmos tenga la misma nomenclatura.

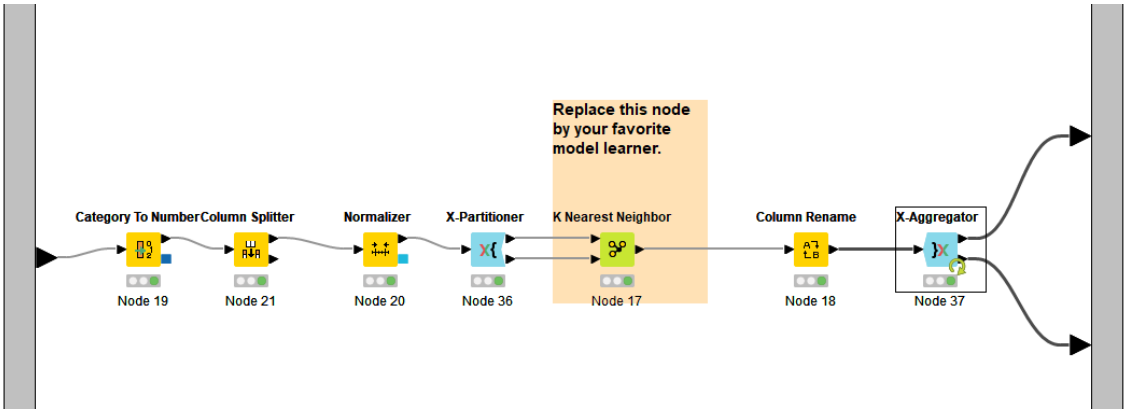


*Figura 14: 5-NN Workflow*

En la Tabla 9 observamos las matrices de confusión conseguidas a partir de este algoritmo

*Tabla 9: Matrices de confusión 5-NN*

| | "No" | "Yes" |
|---|---|---|
| "No" | 32194 | 11122 |
| "Yes" | 3051 | 35993 |

| | "Dropout" | "Graduate" |
|---|---|---|
| "Dropout" | 2051 | 597 |
| "Graduate" | 368 | 2280 |

| | "C" | "A" | "B" | "D" |
|---|---|---|---|---|
| "C" | 1742 | 507 | 751 | 349 |
| "A" | 275 | 2432 | 612 | 29 |
| "B" | 864 | 1138 | 1172 | 173 |
| "D" | 839 | 113 | 257 | 2140 |

Algunas de la medidas de precisión que han sido calculadas se pueden observar en la Tabla 10.

*Tabla 10: Medidas de precisión 5-NN*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Heart | 35993 | 11122 | 32194 | 3051 | 0.92186 | 0.74324 | 0.76394 | 0.82791 | 0.8355 | 1.29038 | 0.89215 |
| Body | 2432 | 1758 | 8287 | 916 | 0.7264 | 0.82499 | 0.58043 | 0.80034 | 0.64526 | 1.24555 | |
| Stud. | 2280 | 597 | 2051 | 368 | 86103 | 0.77455 | 0.79249 | 0.71779 | 0.82534 | 1.2789 | 0.89133 |

## Random Forest

Es un multiclasificador que combina varios clasificadores simples para tratar de mejorar la clasificación. Es un ejemplo de *bagging* que consiste en que cada modelo se induce independientemente sin información de los demás. Lo que busca es mejorar algoritmos inestables que frente pequeños cambios en el conjunto de entrenamiento pueden provocar grandes cambios en la predicción.

Random Forest realiza distintas clasificaciones con árboles más débiles y con diferentes subconjuntos de datos. En la Figura 15 observamos que primero es necesario saber correctamente el dominio de nuestro problema, para ello utilizamos el nodo *Domain Calculator*. Posteriormente añadimos los nodos para la validación cruzada y los nodos correspondientes al modelo y al predictor que utiliza Random Forest. En este caso he optado por utilizar *Information Gain Ratio*, para probar otro tipo de árboles, con 100 modelos.

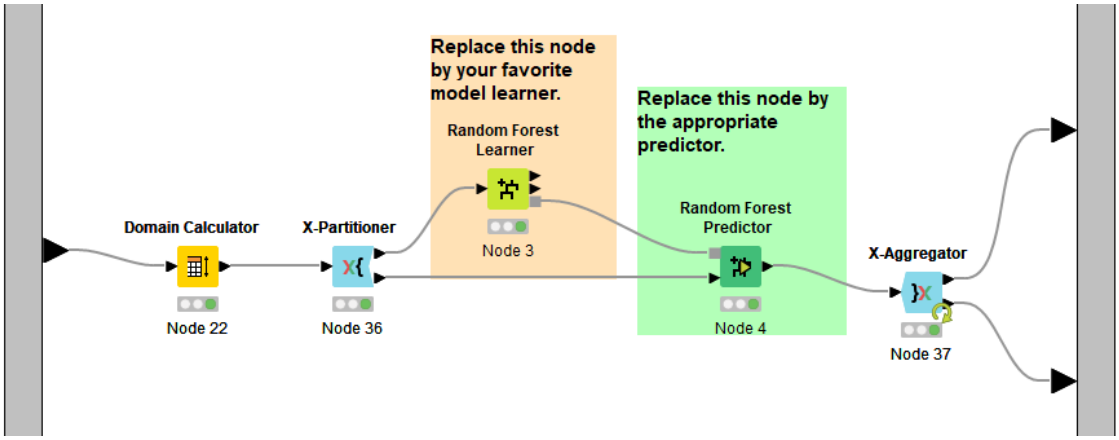


*Figura 15: Random Forest Workflow*

En la Tabla 11 se pueden ver las matrices de confusión del algoritmo en cada problema.

*Tabla 11: Matrices de confusión Random Forest*

| | "No" | "Yes" |
|---|---|---|
| "No" | 37502 | 5814 |
| "Yes" | 2828 | 36216 |

| | "Dropout" | "Graduate" |
|---|---|---|
| "Dropout" | 2398 | 250 |
| "Graduate" | 112 | 2536 |

| | "C" | "A" | "B" | "D" |
|---|---|---|---|---|
| "C" | 2217 | 282 | 659 | 191 |
| "A" | 36 | 2895 | 409 | 8 |

| | | | | |
|---|---|---|---|---|
| “B” | 416 | 772 | 2036 | 123 |
| “D” | 405 | 43 | 175 | 2726 |

Por último, algunas medidas de precisión se pueden observar en la tabla Tabla 12.

*Tabla 12: Medidas de precisión Random Forest*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Heart | 36216 | 5814 | 37502 | 2828 | 0.2757 | 0.96578 | 0.96167 | 0.89507 | 0.89341 | 1.33916 | 0.96583 |
| Body | 2895 | 1097 | 5948 | 453 | 0.8647 | 0.89079 | 0.7252 | 0.88427 | 0.78883 | 1.32495 | |
| Stud. | 2536 | 250 | 2398 | 112 | 0.957704 | 0.905589 | 0.910266 | 0.931647 | 0.933382 | 1.365025 | 0.972209 |

## Boosting

Se va a utilizar también un algoritmo de *boosting*, es decir, un multiclasificador donde en cada iteración se van teniendo en cuenta los fallos del anterior. Para ello, tenemos el nodo *XGBoost Tree Ensemble Learner* y el nodo *XGBoost Predictor*. Pero en aquellos problemas con variables categóricas es necesario que se pasen a variables discretas para un buen funcionamiento del clasificador.

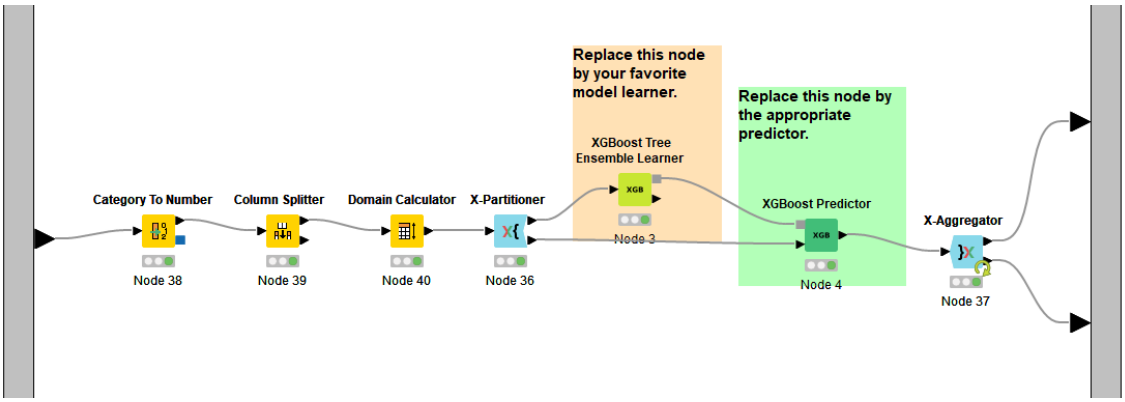


*Figura 16: XGBoost Tree Workflow*

Podemos observar las tasas de aciertos, en la Tabla 13, de cada uno de los problemas.

*Tabla 13: Matrices de confusion XGBoost Tree*

| | “No” | “Yes” |
|---|---|---|
| “No” | 37339 | 5977 |
| “Yes” | 6603 | 32441 |

| | “Dropout” | “Graduate” |
|---|---|---|
| “Dropout” | 2413 | 235 |
| “Graduate” | 117 | 2531 |

| | “C” | “A” | “B” | “D” |
|---|---|---|---|---|
| “C” | 1319 | 402 | 599 | 1029 |
| “A” | 128 | 2809 | 404 | 7 |
| “B” | 919 | 1123 | 1043 | 262 |
| “D” | 337 | 66 | 163 | 2783 |

Las medidas de precisión obtenidas para este algoritmo son las detalladas en al Tabla 14

*Tabla 14: Medidas de precisión XGBoost Tree*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Heart | 32441 | 5977 | 37339 | 6603 | 0.83088 | 0.86201 | 0.84442 | 0.84726 | 0.8376 | 1.30111 | 0.93139 |
| Body | 2809 | 1591 | 8454 | 539 | 0.83901 | 0.84161 | 0.63841 | 0.84096 | 0.72509 | 1.29639 | |
| Stud. | 2531 | 235 | 2413 | 117 | 0.95582 | 0.91125 | 0.91504 | 0.93353 | 0.93498 | 1.36641 | 0.97183 |

# 4. Configuración de algoritmos

En el problema de *Students* se han realizado pequeñas modificaciones a los parámetros de los algoritmos para ver como varía a la hora de crear el modelo y su precisión en las predicciones. Se han cogido los siguientes algoritmos para ver si el peor de todos, k-NN, es capaz de alcanzar a los demás o acercarse a las medidas de precisión obtenidas.

## XGBoost Tree

En este algoritmo se ha optado por aumentar en dos ocasiones el número de iteraciones que se realiza el *boosting*, y obtenemos los resultados de la Tabla 15.

*Tabla 15: Medidas de precisión XGBoost Tree MOD*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| XGB | 2531 | 235 | 2413 | 117 | 0.95582 | 0.91125 | 0.91504 | 0.93353 | 0.93498 | 1.36641 | 0.97183 |
| XGB125 | 2529 | 234 | 2414 | 119 | 0.95506 | 0.91163 | 0.91531 | 0.93335 | 0.93476 | 1.36627 | 0.97209 |
| XGB150 | 2529 | 234 | 2414 | 119 | 0.95506 | 0.9113 | 0.91531 | 0.93335 | 0.93476 | 1.36627 | 0.9721 |

Al aumentar el número de iteraciones hemos conseguido que se reduzca el número de verdaderos positivos pero también que aumente el de verdaderos negativos. Aunque en este algoritmo los cambios no son tan significativos el modelo irá mejorando pues tendrá en cuenta más cambios que se produzcan en versiones anteriores.

## Neural Network

En la red neuronal se han modificado los parámetros correspondientes al número de capas y número de neuronas por capa. La sintaxis de la Tabla 16 queda como *nº capas n nº nueronas nc*.

*Tabla 16: Medidas de precisión Neural Network MOD*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| NN | 2417 | 300 | 2348 | 231 | 0.912764 | 0.886707 | 0.889584 | 0.899736 | 0.901025 | 1.341444 | 0.945225 |
| NN 3c | 2411 | 292 | 2356 | 237 | 0.9105 | 0.88973 | 0.89197 | 0.90011 | 0.90114 | 1.34173 | 0.93763 |
| NN 5c | 2412 | 291 | 2357 | 236 | 0.91088 | 0.89011 | 0.89234 | 0.90049 | 0.90151 | 1.34201 | 0.93739 |
| NN3c25nc | 2431 | 286 | 2362 | 217 | 0.91805 | 089199 | 0.89474 | 0.90502 | 0.90624 | 1.34538 | 0.94976 |
| NN5c20nc | 2419 | 303 | 2345 | 229 | 0.91352 | 0.88557 | 0.88868 | 0.89955 | 0.90093 | 1.3413 | 0.93686 |

Se puede observar que la precisión de los clasificadores aumenta, aunque con una diferencia pequeña de capas parece ser más relevante el numero de neuronas por capa que el número de capas.

### Random Forest

Este multiclasificador consigue muy buenos resultados, vamos a cambiar algunos parámetros a ver si conseguimos mejorarlos aún más. Para ello hemos modificado el tipo de criterio de división del árbol a *Gini Index*, y también hemos aumentado el número de iteraciones.

*Tabla 17: Medidas de precisión Random Forest MOD*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RF | 2536 | 250 | 2398 | 112 | 0.9577 | 0.90559 | 0.91027 | 0.93165 | 0.3338 | 1.36502 | 0.97221 |
| RF GI | 2535 | 250 | 239 | 113 | 0.95733 | 0.90559 | 0.91023 | 0.93146 | 0.93319 | 1.36489 | 0.97195 |
| RF150 | 2537 | 245 | 2403 | 111 | 0.95808 | 0.90748 | 0.91193 | 0.93278 | 0.93444 | 1.36585 | 0.97244 |

Se puede ver en la Tabla 17, que la precisión aumenta cuando aumentan las iteraciones, pero disminuye con el cambio de criterio de división.

### K-NN

En el algoritmo de k-NN se ha optado por aumentar y disminuir el número de vecinos y con el mismo número que hemos realizado el estudio, añadirles peso a las distancias, por lo que los más cercanos tendrán un peso mayor que los más alejados.

*Tabla 18: Medidas de precisión K-NN MOD*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5NN | 2280 | 597 | 2051 | 368 | 0.86103 | 0.77455 | 0.79249 | 0.81779 | 0.82534 | 1.2789 | 0.89133 |
| 3NN | 2215 | 551 | 2097 | 433 | 0.83648 | 0.79192 | 0.8008 | 0.8142 | 0.81825 | 1.27609 | 0.88142 |
| 7NN | 2306 | 612 | 2036 | 342 | 0.87085 | 0.76888 | 0.79027 | 0.81986 | 0.8286 | 1.28052 | 0.8952 |
| 10NN | 2346 | 643 | 2005 | 302 | 0.88595 | 0.75718 | 0.78488 | 0.82156 | 0.83236 | 1.28185 | 0.89515 |
| 5NNwd | 2285 | 491 | 2157 | 363 | 0.86292 | 0.81458 | 0.82313 | 0.83875 | 0.84255 | 1.29518 | 0.90972 |

Por lo que se observa en la Tabla 18 todos los algoritmos mejoran con el aumento del número de vecinos. De hecho, con el k inicial y añadiendo pesos se consigue el mejor valor de precisión del clasificador.

## 5. Análisis de resultados

A la hora de analizar los resultados de los algoritmos de cada problema uno se pregunta, ¿cómo los compara? ¿Qué es más importante, la tasa de aciertos o que las falsas predicciones se reduzcan? Para ello será necesario hacer una tabla resumen con todas las medidas de precisión calculadas y a partir de ellas obtener graficas que nos faciliten entender que está haciendo cada algoritmo en su predicción.

### Heart_2020

En este primer problema compararemos los algoritmos que aparecen en la tabla Tabla 19, y donde aparecen todas las medidas de precisión estudiadas en todo el trabajo.

*Tabla 19: Medidas de precisión Heart*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ZeroR | 0 | 0 | 43316 | 39044 | 0.0 | 1.0 | ? | 0.525935 | ? | 1.0 | 0.499981 |
| NB | 25065 | 8455 | 34861 | 13979 | 0.64197 | 0.80481 | 0.74776 | 0.72761 | 0.69084 | 1.20282 | 0.8175 |
| DT | 35259 | 7011 | 36305 | 3785 | 0.90306 | 0.83814 | 0.83414 | 0.86892 | 0.86723 | 1.31955 | 0.89105 |
| 5-NN | 35993 | 11122 | 32194 | 3051 | 0.92186 | 0.74324 | 0.76394 | 0.82791 | 0.8355 | 1.29038 | 0.89215 |
| RF | 36216 | 5814 | 37502 | 2828 | 0.2757 | 0.96578 | 0.96167 | 0.89507 | 0.89341 | 1.33916 | 0.96583 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| XGB | 32441 | 5977 | 37339 | 6603 | 0.83088 | 0.86201 | 0.84442 | 0.84726 | 0.8376 | 1.30111 | 0.93139 |

Primero de todo observaremos sus matrices de confusión, Figura 17. Aquí observaremos, a partir de la clase positiva “Yes”, como según el algoritmo van variando los aciertos y fallos.

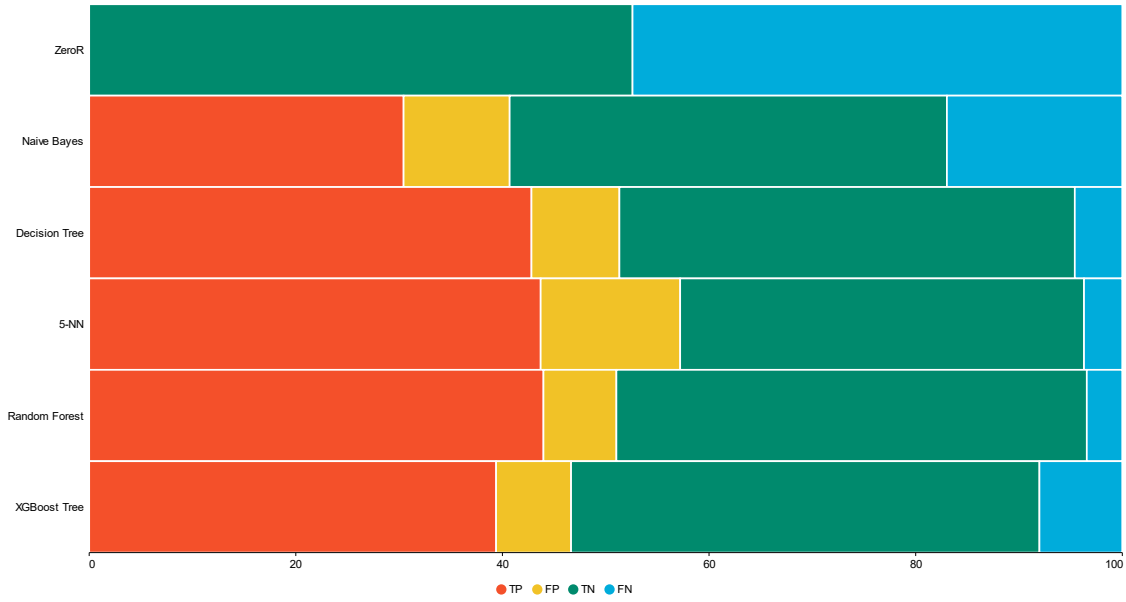


*Figura 17: Tasas de acierto y error Heart*

Buscamos aquellos algoritmos que sean capaces de predecir correctamente los sujetos que padecerán un ataque al corazón, es decir, que el valor de TP, verdaderos positivos, sea muy alta. Los valores más altos son conseguidos por 5-NN, Random Forest y Decision Tree, pero ahora es importante tener en cuenta que la tasa de falsos positivos sea baja, FP. Por lo tanto, 5-NN intenta predecir erróneamente la clase positiva. Por lo tanto en esta primera pasada los algoritmos de Decision Tree y Random Forest se amoldan bastante bien al problema.

Pasamos ahora a comparar la precisión de los algoritmos, para ello observamos la Figura 18

En este primer problema ningún algoritmo supera la cota de 0.9 de precisión, de hecho, es el RF el más próximo a este valor con un 0.895. Sin embargo, son 4 los que consiguen superar el umbral de 0.8, donde se une ahora también el algoritmo de XGBoost Tree, que anteriormente no se tuvo en cuenta por no ser tan alta su tasa de verdaderos positivos, luego también será alta su tasa de verdaderos negativos por lo que su precisión supera a la del algoritmo 5-NN.

Sabemos que en problemas complejos los algoritmos aumentan el número de aciertos de la clase positiva a costa de aumenta el número de errores de dicha clase. Por ello es importante buscar uno que aumente los verdaderos positivos a un ritmo mayor que los falsos positivos. Para encontrarlos utilizamos las curvas ROC y el índice AUC, cuanto más cerca este de 1 más nos acercaremos al clasificador buscado, Figura 19

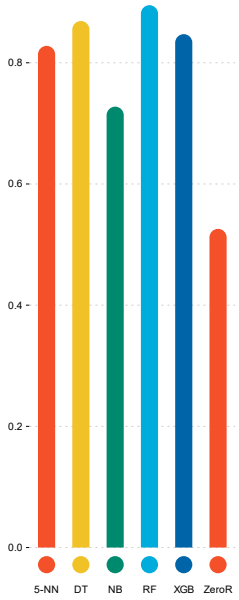


*Figura 18: Precisión algoritmos Heart*

.

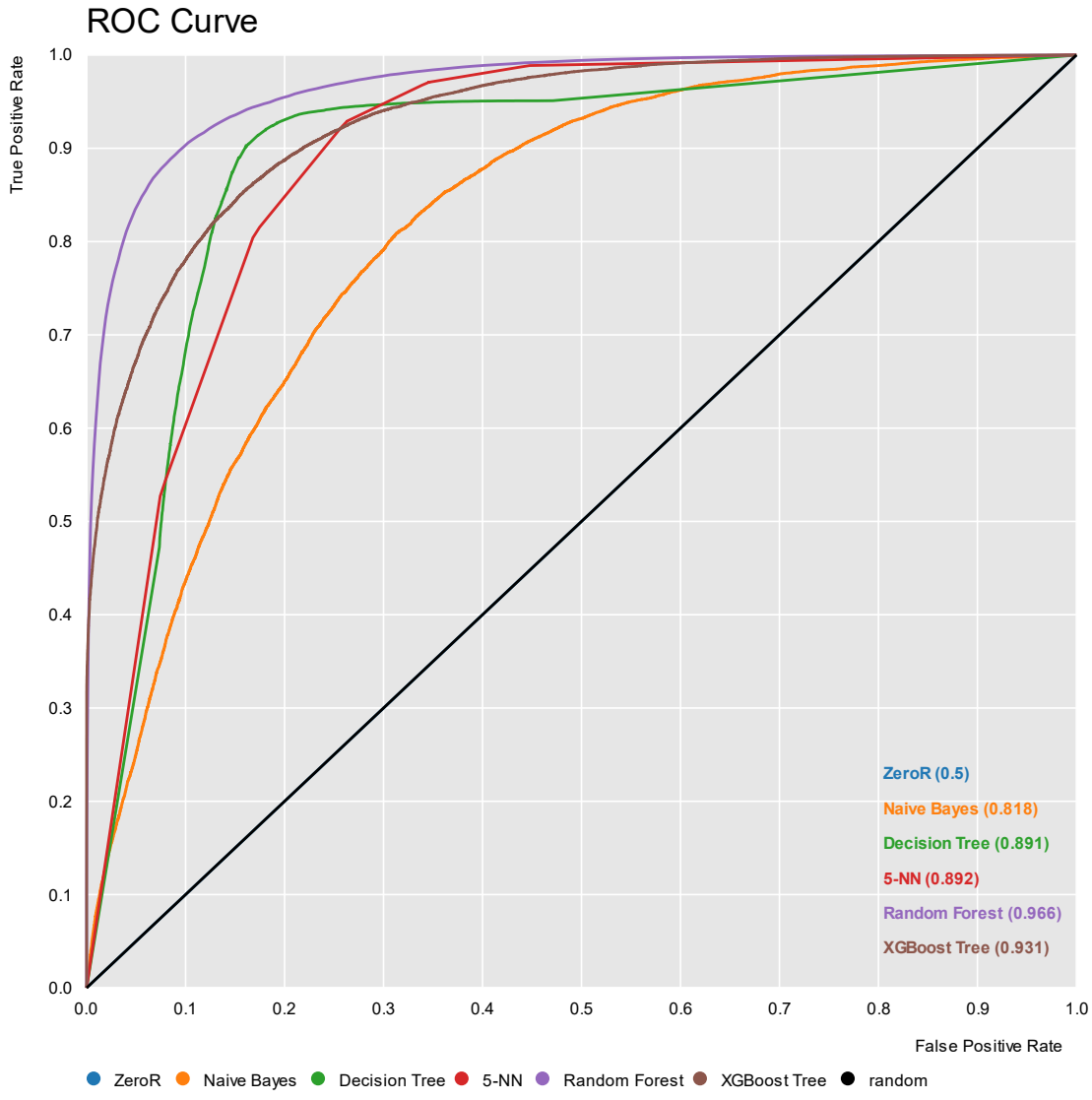

*Figura 19: Curvas ROC Heart*

Para empezar, el algoritmo de Naive Bayes no ha destacado en ningún momento, lo más seguro es que presuponer que todos los atributos de este problema son independientes es bastante erróneo. Por ejemplo, normalmente una persona que hace deporte y no fuma tendrá mejor BMI que uno que no lo haga y si fume. Así existirán más atributos correlacionados, luego es normal que este algoritmo no destaque.

Por otro lado, los clasificadores mas próximos a 1 son Random Forest y XGBoost Tree, es normal que los sean los mejores los multiclasificadores que los clasificadores simples. Aunque no se alejan tanto del Decision Tree.

Por último 5-NN presenta buenos resultados, es posible que falle en instancias de la frontera, ya que como hemos visto en el apartado anterior, modificando sus parámetros y dando peso a las distancias se puede mejorar el algoritmo.

Cabe mencionar que el algoritmo ZeroR coincide en su curva ROC con la de un clasificador aleatorio, luego no supone ninguna diferencia en este aspecto etiquetar a todos de la misma manera o aleatoriamente.

## BodyPerformance

En el problema de multiclase se ha elaborado también la tabla resumen, Tabla 20

, aunque en este caso estamos utilizando el OVA y tomando la clase positiva “A” frente a todas las demás. Vamos a realizar algunas comparaciones con las mismas estrategias que en el anterior problema.

*Tabla 20: Medidas de precisión BodyPerformance*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ZeroR | 669 | 2010 | 8035 | 2679 | 0.199821 | 0.7999 | 0.24972 | 0.649892 | 0.222001 | 0.999861 | |
| DT | 2432 | 1758 | 8287 | 916 | 0.7264 | 0.82499 | 0.58043 | 0.80034 | 0.64526 | 1.2455 | |
| 5-NN | 2432 | 1758 | 8287 | 916 | 0.7264 | 0.82499 | 0.58043 | 0.80034 | 0.64526 | 1.24555 | |
| NN | 2795 | 1868 | 8177 | 553 | 0.834827 | 0.814037 | 0.5994 | 0.819234 | 0.697791 | 0.697791 | |
| RF | 2895 | 1097 | 5948 | 453 | 0.8647 | 0.89079 | 0.7252 | 0.88427 | 0.78883 | 1.32495 | |
| XGB | 2809 | 1591 | 8454 | 539 | 0.83901 | 0.84161 | 0.63841 | 0.84096 | 0.72509 | 1.29639 | |

Calculando la matriz de confusión de todos los clasificadores obtenemos la Figura 20

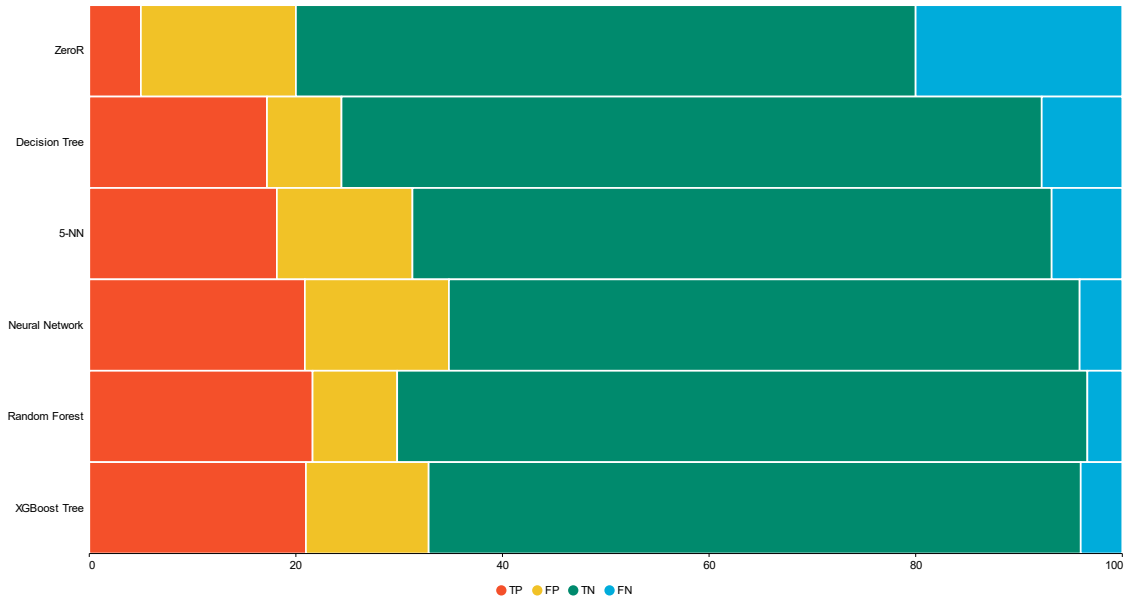


*Figura 20: Tasas de acierto y error BodyPerformance*

Buscamos aquel algoritmo que consiga el mayor número de verdaderos positivos, aunque en problemas multiclase hay que tener en cuenta que existen más de dos clases por lo que es normal que la barra en la gráfica sea tan pequeña aunque el problema este balanceado. Nos encontramos la Neural Network, el Random Forest y el XGBoost Tree. Seguimos consiguiendo mejores resultados en multiclasificadores, aunque la tasa de verdaderos positivos no se aleja tanto de peor de los tres.

Comparando ahora la precisión de los clasificadores gracias a la Figura 21.

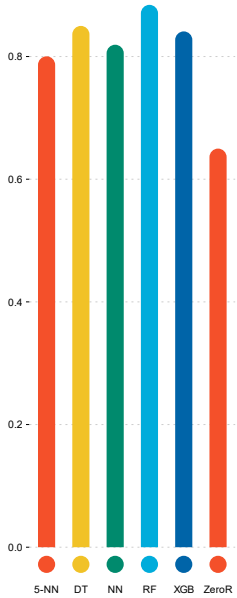


*Figura 21: Precisión algoritmos BodyPerformance*

Todos los algoritmos alcanzan la precisión de 0.8, pero ninguno supera el umbral de 0.9. Observamos que el Decision Tree supera a la Neural Network considerablemente en la precisión de sus predicciones. De nuevo se ve la clara ventaja de los multiclasificadores con respecto a

clasificadores simples, donde el algoritmo de Random Forest supera la precisión de 0.89, por lo que es un buen candidato para tomar como mejor clasificador.

Con respecto a las curvas ROC en problemas multiclasificadores, sería necesario tomar la estrategia OVO y realizar todas las combinaciones binarias posibles de clases positivas y negativas para poder realizar una buena comparación, pero es algo muy costoso y largo de realizar.

## Students

En este último problema, además de los algoritmos estudiados en otros, también tenemos las modificaciones realizadas en el apartado anterior que también serán estudiadas en las curvas ROC, para ver el peso de la modificación en la mejora del clasificador.

*Tabla 21: Medidas de precisión Students*

| | TP | FP | TN | FN | TPR | TNR | PPV | Accur. | F1-score | G-mean | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ZeroR | 1058 | 1060 | 1588 | 1590 | 0.399547 | 0.599698 | 0.499528 | 0.499622 | 0.443978 | 0.999622 | 0.499547 |
| DT | 2351 | 282 | 2366 | 287 | 0.88784 | 0.8935 | 0.8929 | 0.89067 | 0.89036 | 1.33467 | 0.89185 |
| 5-NN | 2280 | 597 | 2051 | 368 | 86103 | 0.77455 | 0.79249 | 0.71779 | 0.82534 | 1.2789 | 0.89133 |
| 3NN | 2215 | 551 | 2097 | 433 | 0.83648 | 0.79192 | 0.8008 | 0.8142 | 0.81825 | 1.27609 | 0.88142 |
| 7NN | 2306 | 612 | 2036 | 342 | 0.87085 | 0.76888 | 0.79027 | 0.81986 | 0.8286 | 1.28052 | 0.8952 |
| 10NN | 2346 | 643 | 2005 | 302 | 0.88595 | 0.75718 | 0.78488 | 0.82156 | 0.83236 | 1.28185 | 0.89515 |
| 5NNwd | 2285 | 491 | 2157 | 363 | 0.86292 | 0.81458 | 0.82313 | 0.83875 | 0.84255 | 1.29518 | 0.90972 |
| NN | 2417 | 300 | 2348 | 231 | 0.912764 | 0.886707 | 0.889584 | 0.899736 | 0.901025 | 1.341444 | 0.945225 |
| NN 3c | 2411 | 292 | 2356 | 237 | 0.9105 | 0.88973 | 0.89197 | 0.90011 | 0.90114 | 1.34173 | 0.93763 |
| NN 5c | 2412 | 291 | 2357 | 236 | 0.91088 | 0.89011 | 0.89234 | 0.90049 | 0.90151 | 1.34201 | 0.93739 |
| NN3c25nc | 2431 | 286 | 2362 | 217 | 0.91805 | 089199 | 0.89474 | 0.90502 | 0.90624 | 1.34538 | 0.94976 |
| NN5c20nc | 2419 | 303 | 2345 | 229 | 0.91352 | 0.88557 | 0.88868 | 0.89955 | 0.90093 | 1.3413 | 0.93686 |
| RF | 2536 | 250 | 2398 | 112 | 0.9577 | 0.90559 | 0.91027 | 0.93165 | 0.3338 | 1.36502 | 0.97221 |
| RF GI | 2535 | 250 | 239 | 113 | 0.95733 | 0.90559 | 0.91023 | 0.93146 | 0.93319 | 1.36489 | 0.97195 |
| RF150 | 2537 | 245 | 2403 | 111 | 0.95808 | 0.90748 | 0.91193 | 0.93278 | 0.93444 | 1.36585 | 0.97244 |
| XGB | 2531 | 235 | 2413 | 117 | 0.95582 | 0.91125 | 0.91504 | 0.93353 | 0.93498 | 1.36641 | 0.97183 |
| XGB125 | 2529 | 234 | 2414 | 119 | 0.95506 | 0.91163 | 0.91531 | 0.93335 | 0.93476 | 1.36627 | 0.97209 |
| XGB150 | 2529 | 234 | 2414 | 119 | 0.95506 | 0.9113 | 0.91531 | 0.93335 | 0.93476 | 1.36627 | 0.9721 |

A partir de la Figura 22 podemos observar la tasa de verdaderos positivos de los clasificadores iniciales.

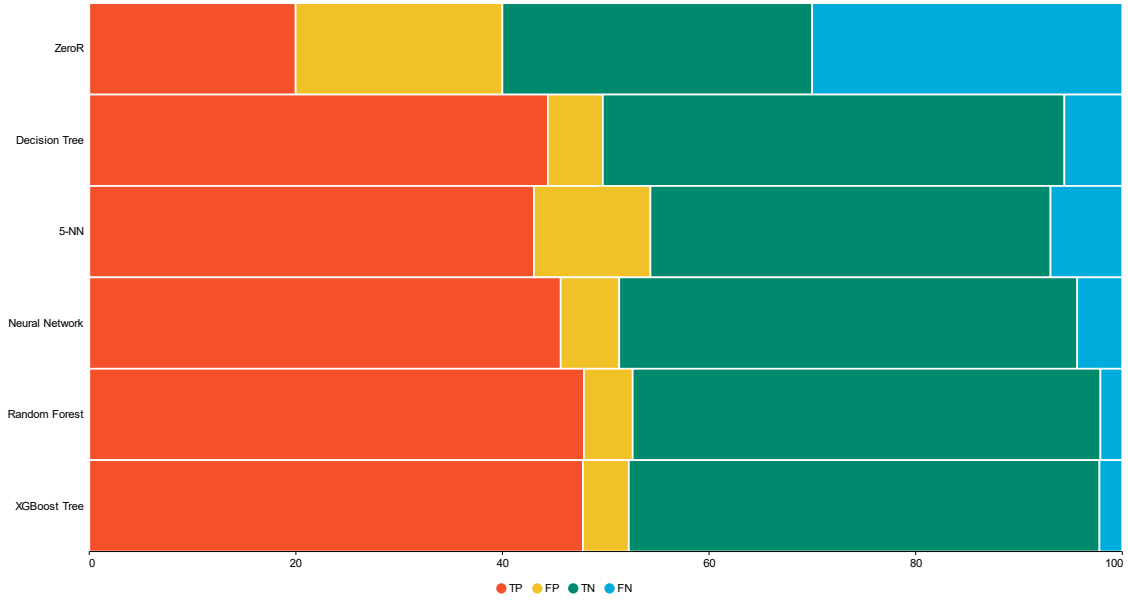


*Figura 22: Tasas de acierto y error Students*

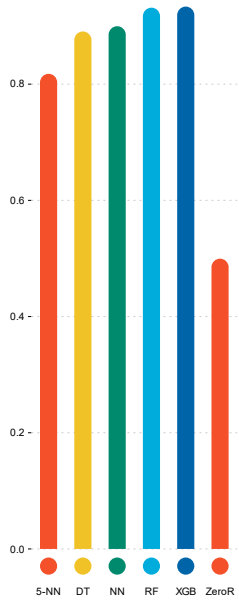


*Figura 23: Precision algoritmos Students*

Podemos observar una gran cantidad de verdaderos positivos y verdaderos negativos, luego parece que el procesamiento de los datos ha sido bastante bueno para todos los algoritmos. Si buscamos el que mayor tasa de verdaderos positivos nos encontramos casi un empate entre los multiclasificadores, veremos en las siguientes comparaciones quien obtiene mejores resultados.

Por otro lado el clasificador de vecinos cercano no consigue acertar tan bien la clase positiva, como hemos visto en el apartado anterior, hubiese sido bastante acertado darles peso a las distancias para mejorar las predicciones.

Pasamos ahora a comparar la precisión de estos mismos clasificadores en la Figura 23.

Seguimos encontrando casi una igualdad en los multiclasificadores que superan con creces el umbral de 0.9, aunque no se plasme en la gráfica. Sin embargo el árbol de decisión y la red

neuronal no se quedan atrás casi alcanzandolo, parece que también eran capaces de predecir instancias de la clase negativa correctamente.

Por último compararemos las curvas ROC de todos los algoritmos estudiados en este problema

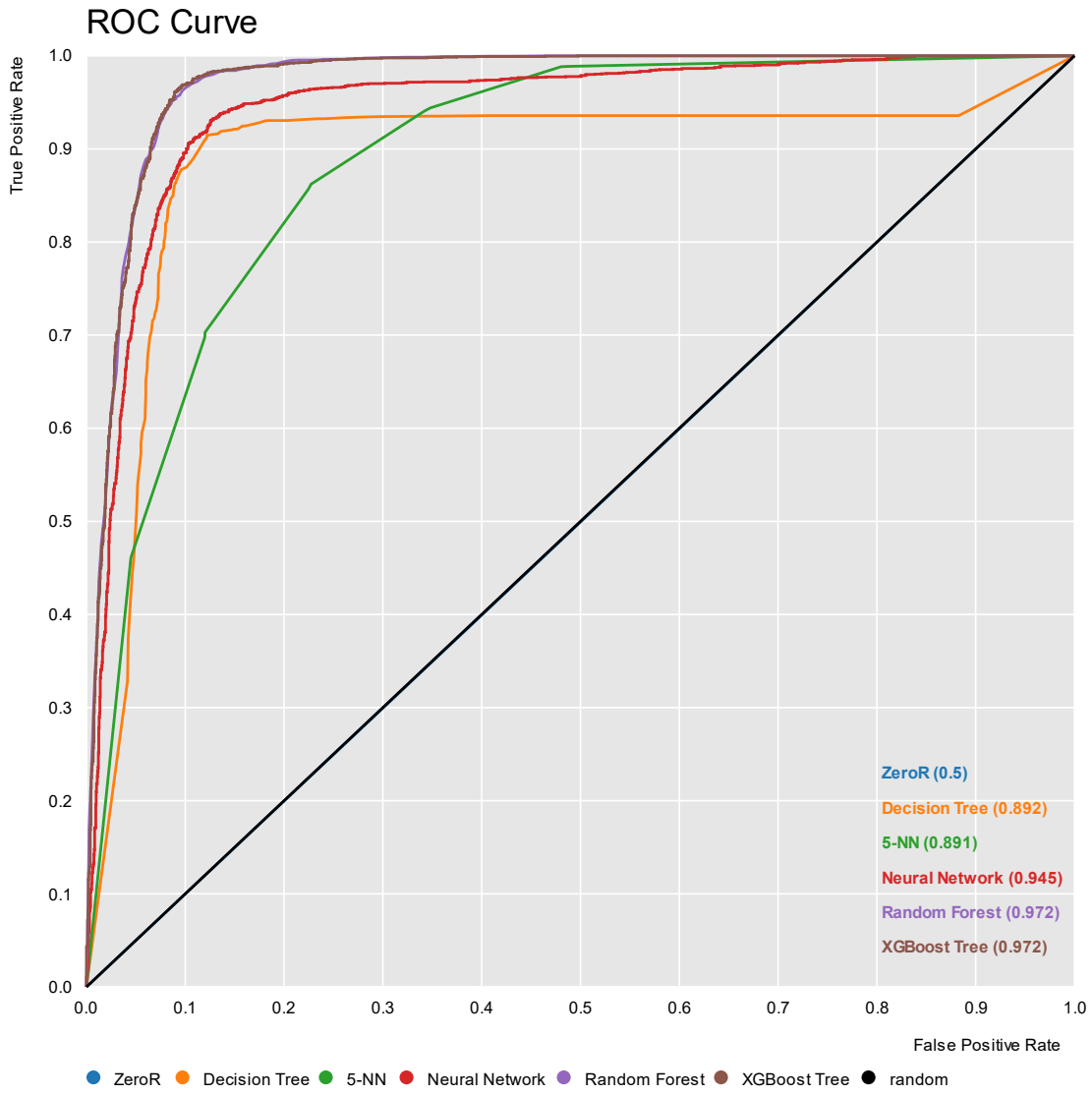


*Figura 24: Curvas ROC Students*

En la Figura 24 se observan los clasificadores iniciales donde los multiclasificadores siguen teniendo al ventaja, de hecho la misma prácticamente. Pero cabe destacar que en el Decision Tree la tasa de verdaderos positivos no aumenta más rápido que la de falsos positivos, por lo que se queda más atrás que la red neuronal. Esto puede ocurrir porque el problema tiene todo valores numéricos, de hecho continuos, y los árboles de decisión trabajan mejor con variables discretas o categóricas si los intervalos no son tan claros.

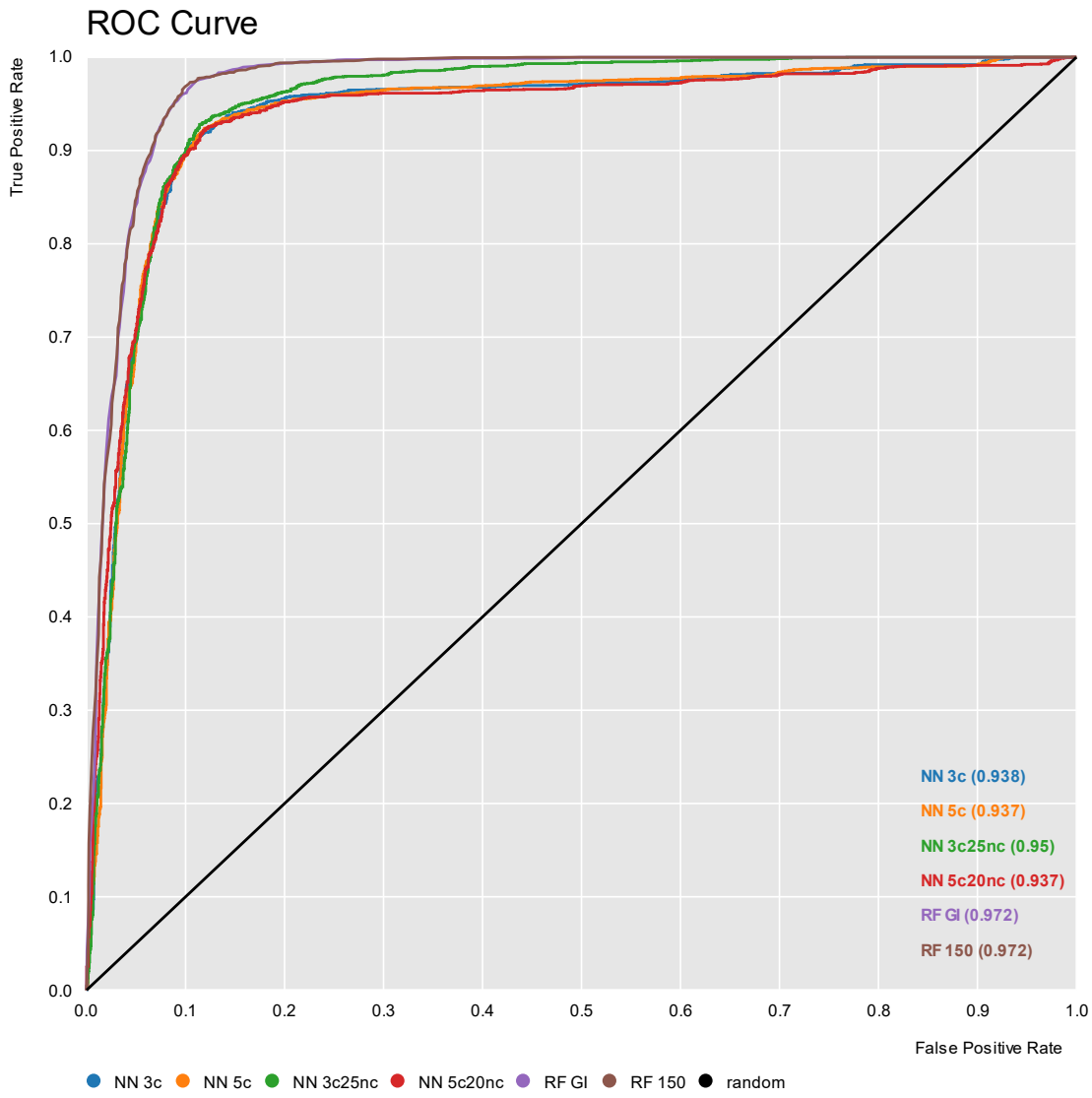


*Figura 25: Curvas ROC Students Mod1*

En la Figura 25 observamos las modificaciones en las redes neuronales y el Random Forest. Hay poco que añadir pues la mejora no es tan significativa, salvo en la red neuronal que contine mayor cantidad de neuronas, ahí si se aumenta la medida AUC. Lo mencionamos anteriormente, si tiene mas neuronas será capaz de dividir el problema en la caja negra mejor.

Por otro lado en la Figura 26tenemos los algoritmos de k-NN y de XGBoost Tree. Los multiclasificadores no consiguen tanta mejora con las modificaciones. Sin embargo, el clasificador de k vecinos cercanos consigue un con cada aumento en el número de vecinos, pero una mayor recompensa si se les añaden pesos a las distancias. ¿Esto porque es así? Pues tiene sentido que aquellas instancias que están muy muy cerca presenten la misma clase aunque la mayoría sea otra.

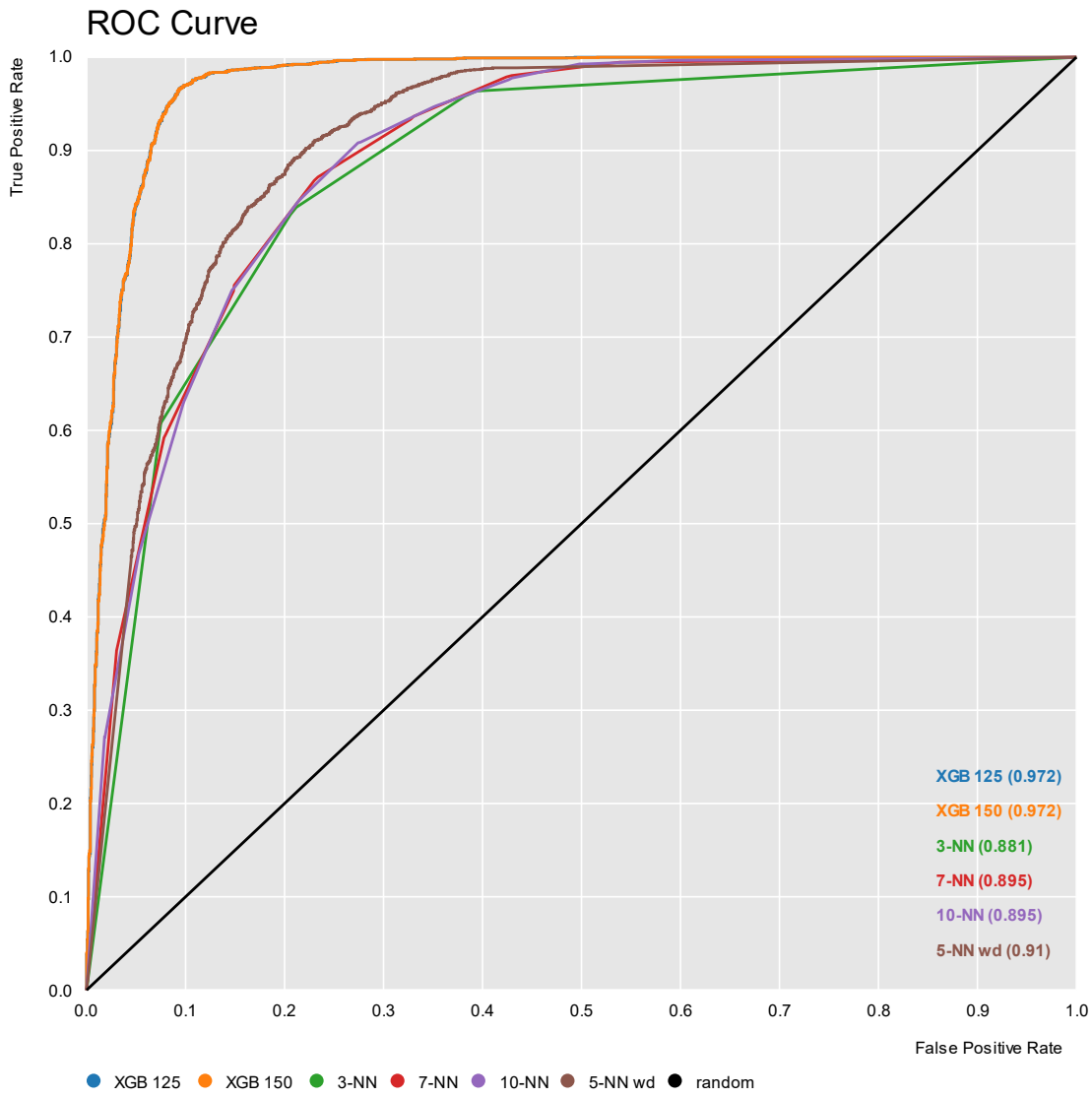


*Figura 26: Curvas ROC Students Mod2*

## 6. Interpretación de los datos

Vamos a realizar una interpretación de los datos del problema de Students. Para ello partimos de la última iteración del algoritmo de Random Forest y visualizamos el árbol obtenido, Figura 27.

Vemos que la primera división la realiza en torno a la variable *Previous qualification* y la separa en mayor o menor que 1.0095. La siguiente división se basa en la variable *Curricular units 1st sem (approved)* y en cada rama realiza una división con respecto a un valor distinto. Por otro lado, aparece la variable *Father's occupation* para dividir una de las ramas de nuevo.

Tiene sentido que estas variables sean las primeras en aparecer, pues un alumno tendrá más interés en aprobar y graduarse si sus notas son buenas o si en cada semestre va aprobando asignaturas.

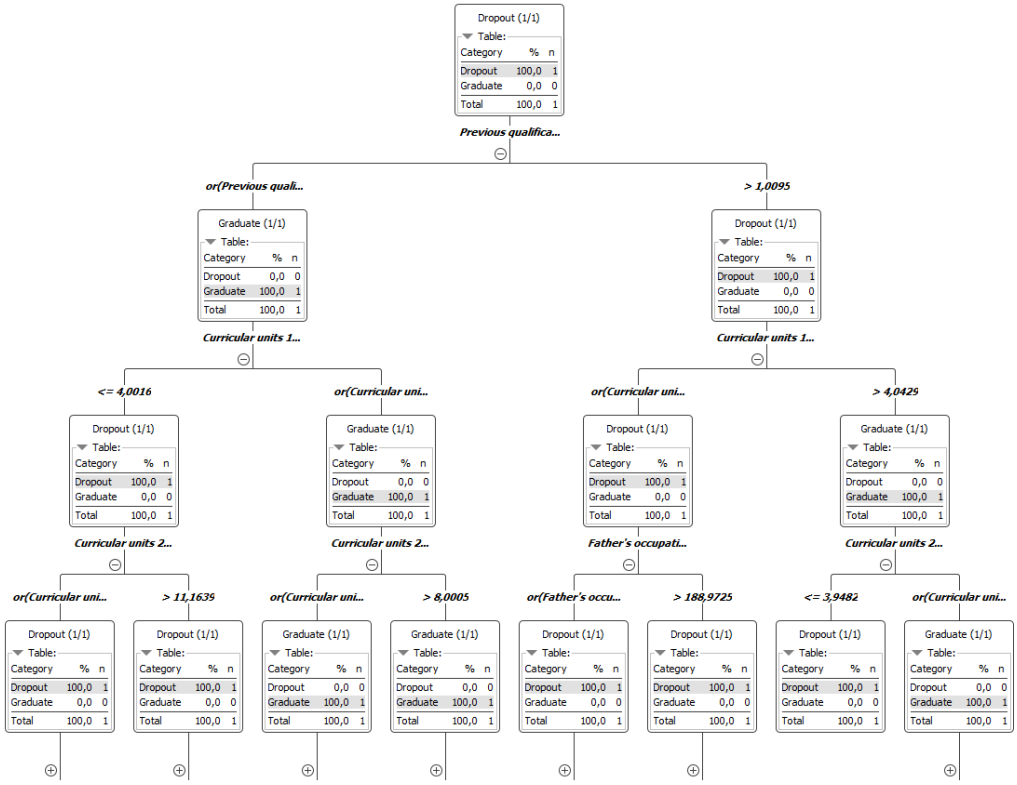


*Figura 27: Árbol de decisión Random Forest*

Veamos ahora la Tabla 22 con la cantidad de veces que ha sido seleccionado un atributo para dividir el árbol en cada nivel. Vemos que *Curricular units 2nd sem (approved)* tiene gran relevancia en las divisiones. Como ya hemos mencionado también las del primer semestres son importantes, no a tan gran escala, pero si se utilizan bastantes. Esto tiene sentido ya que si al final de un año educativo el alumno no consigue sus objetivos lo mas seguro es que pierda el interés y finalmente deje la carrera.

*Tabla 22: Atributos seleccionados para la división - RandomForest*

| Atributo | #splits (level 0) | #splits (level 1) | #splits (level 2) |
|---|---|---|---|
| Marital status | 0 | 0 | 2 |
| Application mode | 0 | 3 | 5 |
| Application order | 0 | 1 | 4 |
| Course | 1 | 7 | 14 |
| Daytime/evening attendance | 0 | 0 | 1 |
| Previous qualification | 1 | 3 | 5 |
| Previous qualification (grade) | 0 | 2 | 6 |
| Nacionality | 4 | 7 | 13 |
| Mother's qualification | 0 | 3 | 7 |
| Father's qualification | 0 | 3 | 6 |
| Mother's occupation | 0 | 5 | 8 |
| Father's occupation | 0 | 2 | 5 |
| Admission grade | 0 | 5 | 5 |
| Displaced | 0 | 0 | 2 |
| Educational special needs | 0 | 0 | 4 |

| | | | |
|---|---|---|---|
| Debtor | 7 | 9 | 10 |
| Tuition fees up to date | 10 | 21 | 23 |
| Gender | 0 | 0 | 7 |
| Scholarship holder | 6 | 7 | 15 |
| Age at enrollment | 1 | 2 | 15 |
| International | 0 | 0 | 5 |
| Curricular units 1st sem (credited) | 0 | 4 | 12 |
| Curricular units 1st sem (enrolled) | 4 | 6 | 12 |
| Curricular units 1st sem (evaluations) | 4 | 5 | 16 |
| Curricular units 1st sem (approved) | 10 | 14 | 34 |
| Curricular units 1st sem (grade) | 13 | 7 | 16 |
| Curricular units 1st sem (without evaluations) | 0 | 1 | 5 |
| Curricular units 2nd sem (credited) | 1 | 5 | 9 |
| Curricular units 2nd sem (enrolled) | 2 | 12 | 14 |
| Curricular units 2nd sem (evaluations) | 3 | 6 | 11 |
| Curricular units 2nd sem (approved) | 20 | 28 | 44 |
| Curricular units 2nd sem (grade) | 12 | 23 | 22 |
| Curricular units 2nd sem (without evaluations) | 1 | 3 | 2 |
| Unemployment rate | 0 | 0 | 1 |
| Inflation rate | 0 | 0 | 1 |
| GDP | 0 | 1 | 3 |

# 7. Contenido adicional

# 8. Bibliografía


https://www.kaggle.com/datasets/kukuroo3/body-performance-data

https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease

https://plotdb.com/chart/

https://ccia.ugr.es/~casillas/knime.html