ANÁLISIS PREDICTIVO EMPRESARIAL MEDIANTE CLASIFICACIÓN

Práctica1

Inteligencia de Negocio

Ismael Sánchez García



ÍNDICE

[1. INTRODUCCIÓN 2](#_Toc497419422)

[2. RESULTADOS OBTENIDOS 3](#_Toc497419423)

[2.1.KNN 9](#_Toc497419424)

[2.2.C4.5 10](#_Toc497419425)

[2.3. LOGISTIC REGRESSION 11](#_Toc497419426)

[2.4. RANDOM FOREST 12](#_Toc497419427)

[2.5. GRADIENT BOOSTED 13](#_Toc497419428)

[2.6. NETURAL NETWORKS 14](#_Toc497419429)

[3.ANÁLISIS DE RESULTADOS 15](#_Toc497419430)

[4.CONFIGURACIÓN DE ALGORITMOS 17](#_Toc497419431)

[4.1. DECISION TREE (C4.5) 17](#_Toc497419432)

[RESULTADOS 17](#_Toc497419433)

[4.2. NEURAL NETWORKS 19](#_Toc497419434)

[RESULTADOS 19](#_Toc497419435)

[RESULTADOS 21](#_Toc497419436)

[5.PROCESADO DE DATOS 22](#_Toc497419437)

[5.1.ONE TO MANY 22](#_Toc497419438)

[RESULTADOS 22](#_Toc497419439)

[5.3. CORRELACIÓN 24](#_Toc497419440)

[RESULTADOS 25](#_Toc497419441)

[5.4. NORMALIZAR 27](#_Toc497419442)

[RESULTADOS 27](#_Toc497419443)

[5.4. EQUILIBRAR CLASES 31](#_Toc497419444)

[RESULTADOS 31](#_Toc497419445)

[6. INTERPRETACIÓN DE RESULTADOS 33](#_Toc497419446)

[7. CONTENIDO ADICIONAL 35](#_Toc497419447)

[7.1. SUBMUESTREO 35](#_Toc497419448)

[RESULTADOS 37](#_Toc497419449)

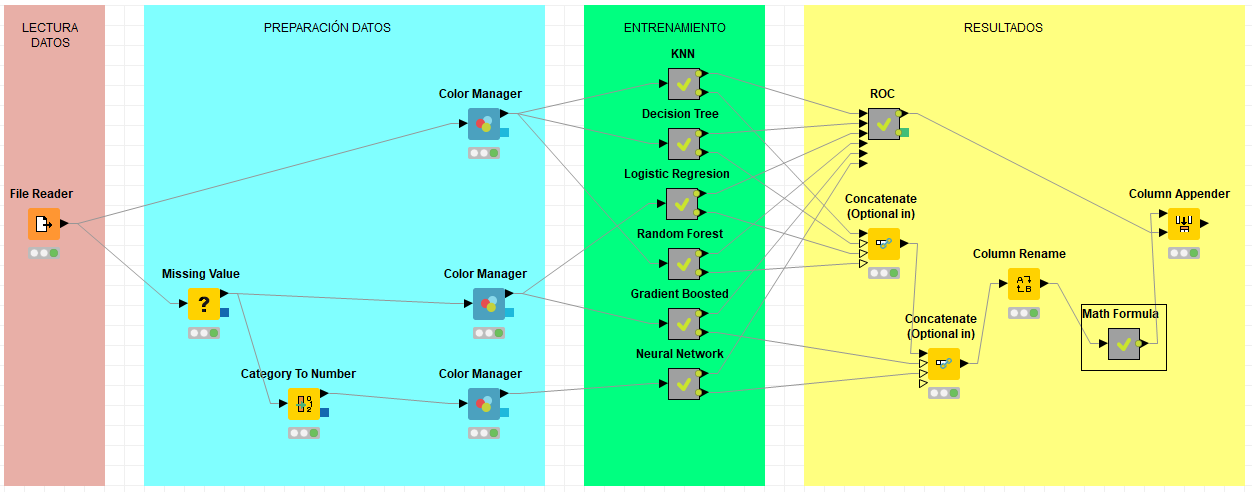
[8.BIBLIOGRAFÍA 41](#_Toc497419450)

# INTRODUCCIÓN

* El problema que vamos a abordar consiste en el análisis de ingresos basado en datos socioeconómicos:
  + El conjunto de datos utilizado es *Adult, que usaremos* para estudiar la influencia de determinados factores demográficos y socioeconómicos sobre la capacidad de ingresos anuales de una persona­.
  + Los datos son casos reales extraídos del censo de EE.UU.
  + El conjunto de datos contiene 14 características (6 numéricas y 8 categóricas) y 48.842 ejemplos (algunos de los cuales presenta valores desconocidos) e incluye información sobre la edad, educación, ocupación, sexo, etnia, etc.
  + La tarea de predicción es determinar si una persona gana más de 50.000$ al año.
* Inicialmente vamos a realizar experimentos con 6 algoritmos distintos con los valores por defecto y sin ningún tratamiento previo de los datos, de forma que podamos ir viendo cómo evolucionan los resultados obtenidos. Los algoritmos se han elegido basándonos en el tipo de problema a abordar, teniendo en cuenta que ni siquiera los científicos de datos con más experiencia pueden determinar qué algoritmo funcionará mejor antes de probarlos.
* Las consideraciones previas que he tenido a la hora de elegir un algoritmo son:
  + Precisión
  + Tiempo de entrenamiento
  + Linealidad (algoritmos que suponen que las clases pueden estar separadas mediante una línea recta)
  + Cantidad de parámetros
  + Cantidad de características
* Algoritmos elegidos:
  + **KNN:** Es un algoritmo no lineal, muy simple, que nos da cierta información del “por qué” ha clasificado un ejemplo en una clase concreta siempre que el número de vecinos sea un número bajo. Las características deben ser numéricas y normalizadas.
  + **Árbol de decisión:** Es un algoritmo no lineal, fácil de utilizar y muy eficiente. Su principal característica es que son fáciles de interpretar.
  + **Regresión logística**: Es un algoritmo lineal, con un tiempo de aprendizaje bastante rápido, y el cual es relativamente sencillo de usar, ya que no es necesaria la configuración de muchos parámetros. Al ser un modelo lineal, en caso de que los datos no sean separables linealmente, no obtendrá resultados muy buenos.
  + **Random Forest**: Es un algoritmo no lineal, con una alta precisión y un tiempo de entrenamiento no muy alto
  + **Gradient Boosted:** Es un algoritmo no lineal, sencillo y con un tiempo de aprendizaje rápido (dependiendo del número de niveles).
  + **Red Neuronal:** Es un algoritmo no lineal, con una alta precisión pero que no permite la interpretación de los resultados, requiere de mucho tiempo para el aprendizaje, un alto número de ejemplos y es muy personalizable, por lo que, en muchas ocasiones, se requiere mucho tiempo de prueba.

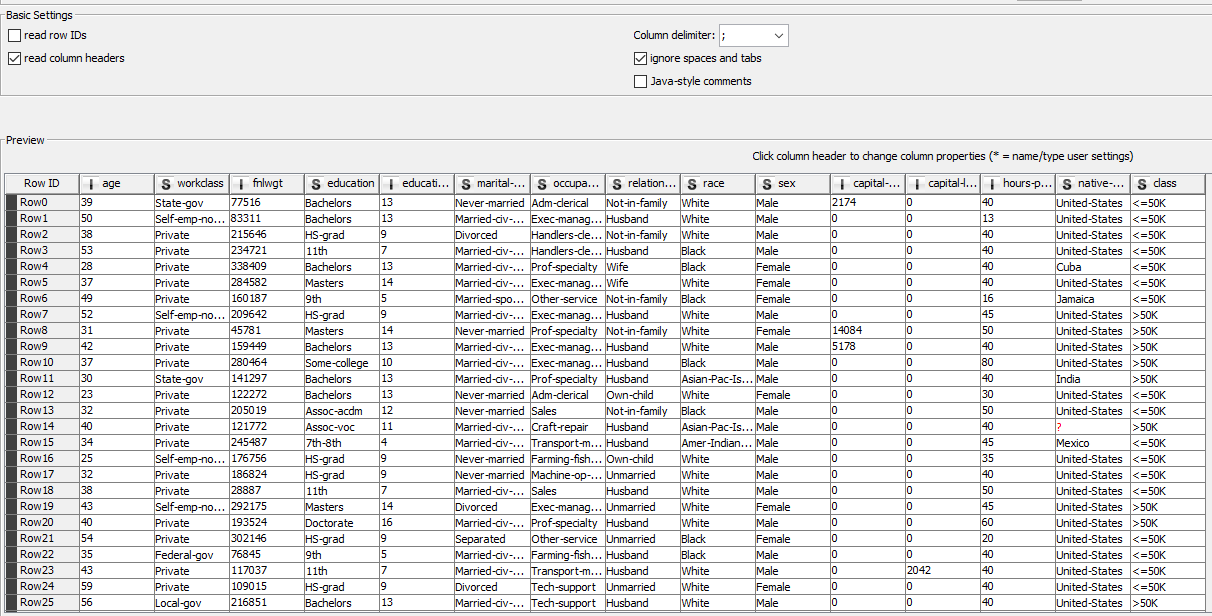
# RESULTADOS OBTENIDOS

* **Flujo de trabajo:**

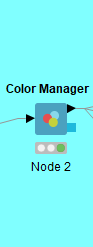
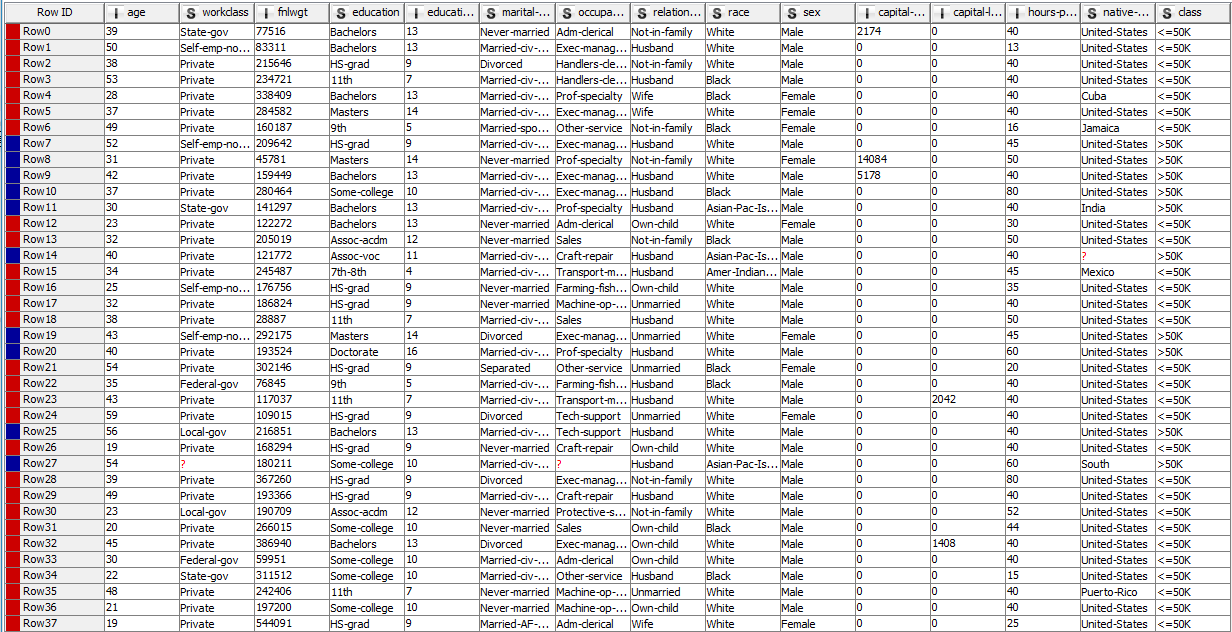


Flujo completo

El flujo de trabajo comienza con la lectura del conjunto de datos:



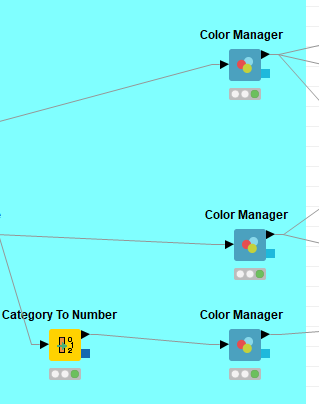
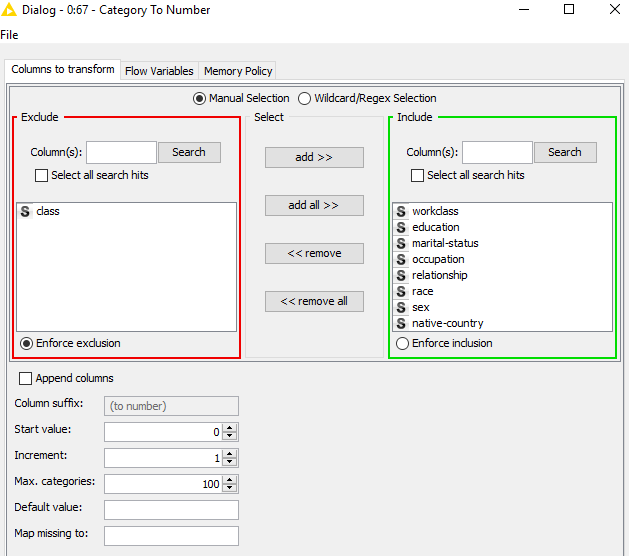
A continuación, le añadimos un color a cada fila según la clase a la que pertenece:



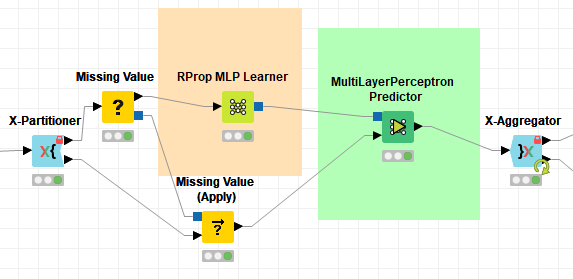
Para los algoritmos “Logistic Regresion”, “Gradien Boosted” y “Neural Network” es necesario tratar los valores perdidos , por lo que he puesto los valores perdidos numéricos como la media y los string como el valor que más común. Dado que los datos de “test” no los conocemos en un problema real a priori, no podemos usarlos para obtener información al hacer esto, por lo tanto, creamos un modelo con los datos de train para tratar los datos perdidos, y con ese modelo tratamos tanto los datos de train como lo de test.

Además de esto, para el algoritmo “Neural Networks” es necesario que todos los valores sean numéricos , por lo que también han sido transformados.

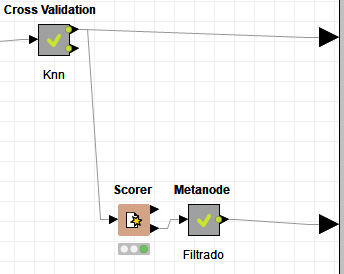
Aquí vemos los 3 flujos distintos:



Aquí podemos observar el tratamiento de los valores perdidos para “Neural Networks”, pero que es igual para “Logistic Regression” y “Gradien Boosted”.

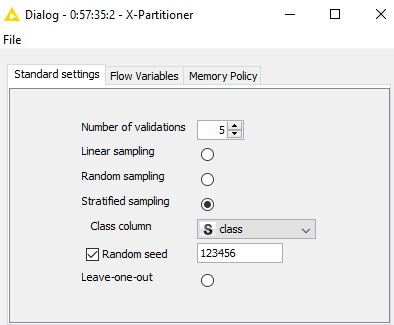


Los siguientes metanodos, en la zona verde, es la aplicación de los distintos algoritmos. Cada metanodo tiene dos salidas, una salida es la tabla de predicción, resultante de aplicar validación cruzada. En la otra salida tenemos la matriz de confusión junto a otros datos de las estadísticas de precisión. Se muestra como ejemplo el metanodo knn, y que es igual en todos los algoritmos:

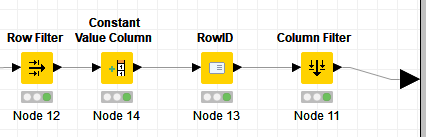


Entrenamiento

Para la validación cruzada, hacemos 5 particiones y ponemos una semilla igual para todos los algoritmos:

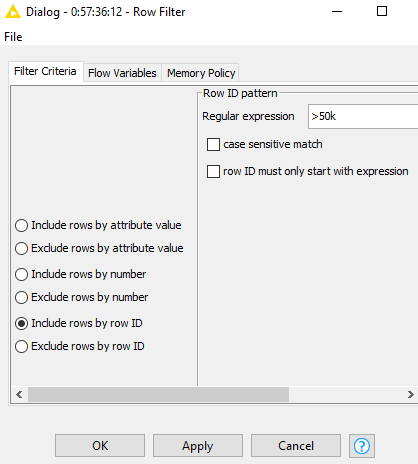


Como podemos ver, dentro hay un metanodo llamado Filtrado en el que hacemos varias operaciones:

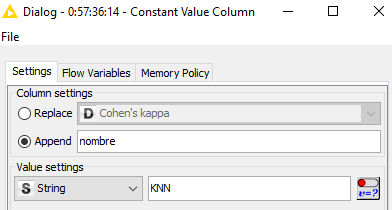


Metanode(Filtrado)

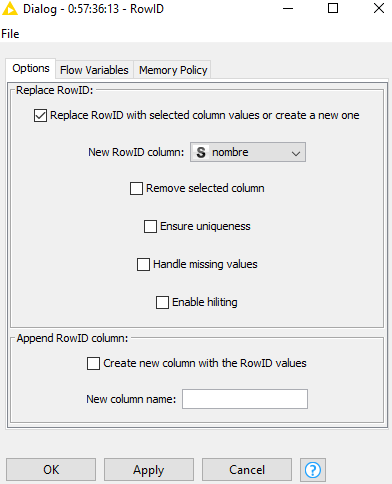
* Filtramos las filas y nos quedamos con la fila con “row ID = >50k”.



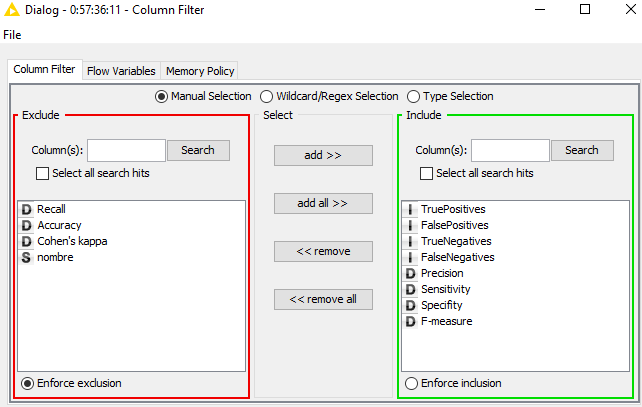
* + Creamos una nueva columna “nombre” con valor el nombre del algoritmo.



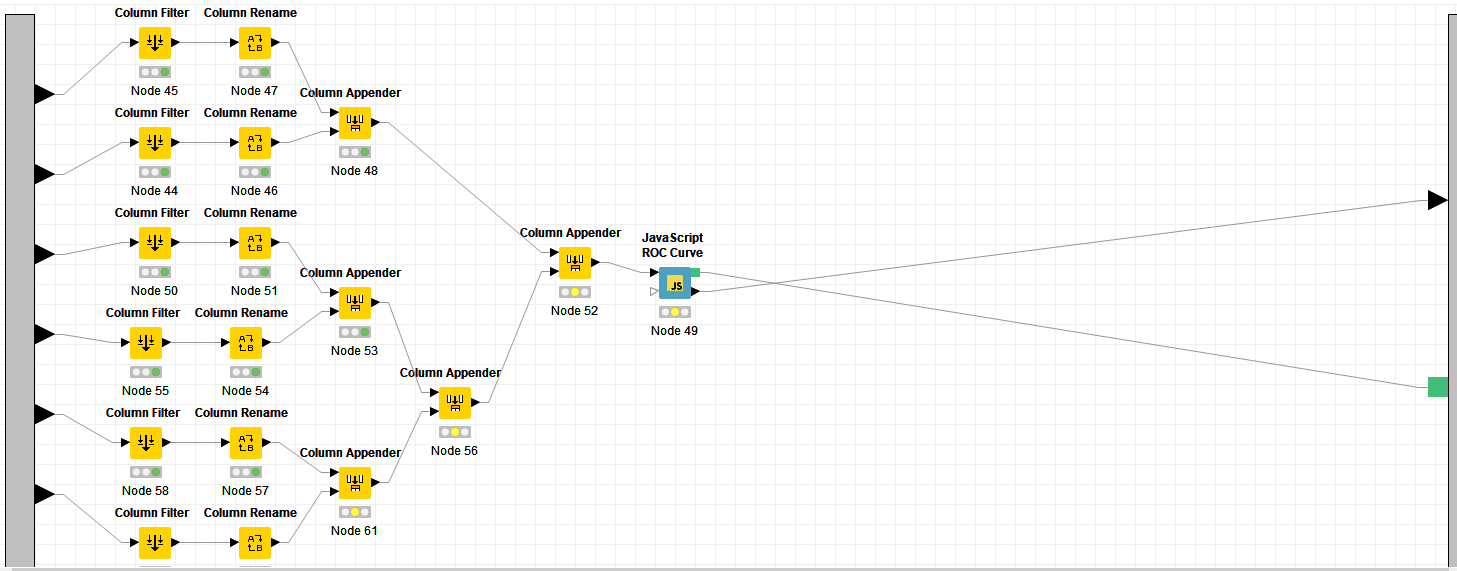
* + Cambiamos el valor de la columna RowID por el de la columna nombre que acabamos de crear.



* + Por último filtramos las columnas, eliminando las que no necesitamos.

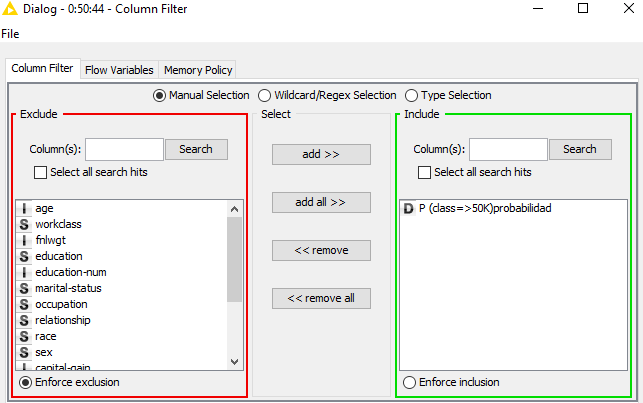


La salida de la validación cruzada (salida superior) la mandamos al nodo ROC(zona amarilla de resultados), en el cual, hacemos las siguientes operaciones:

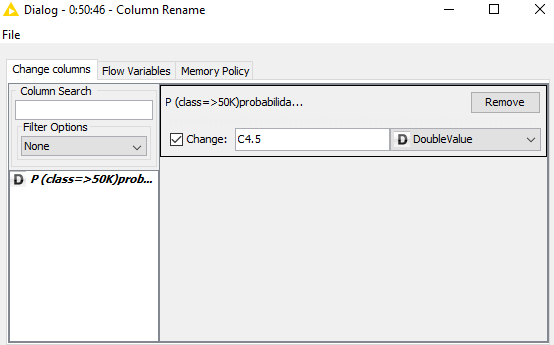


Metanodo (ROC)

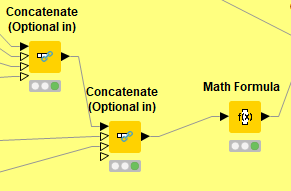
* + Nos quedamos con la columna de probabilidad >50(Column Filter).



* + La renombramos con el nombre de cada algoritmo para obtener la curva ROC(Column Rename).

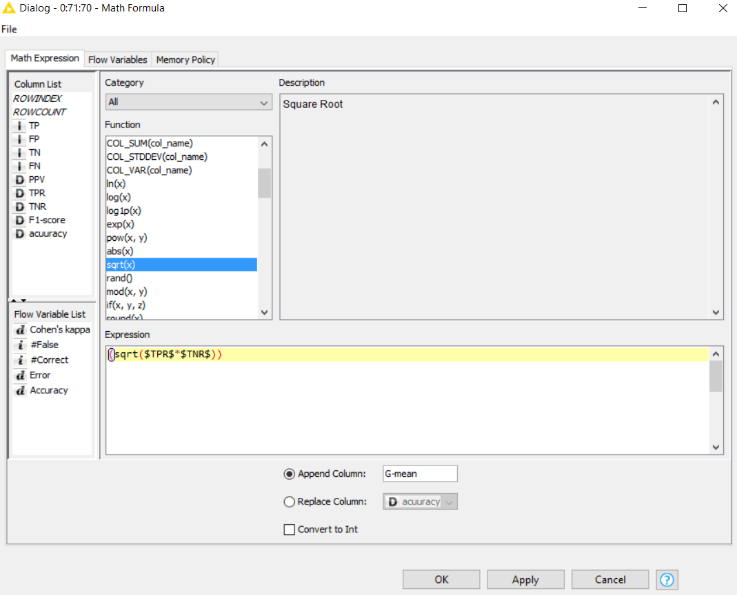
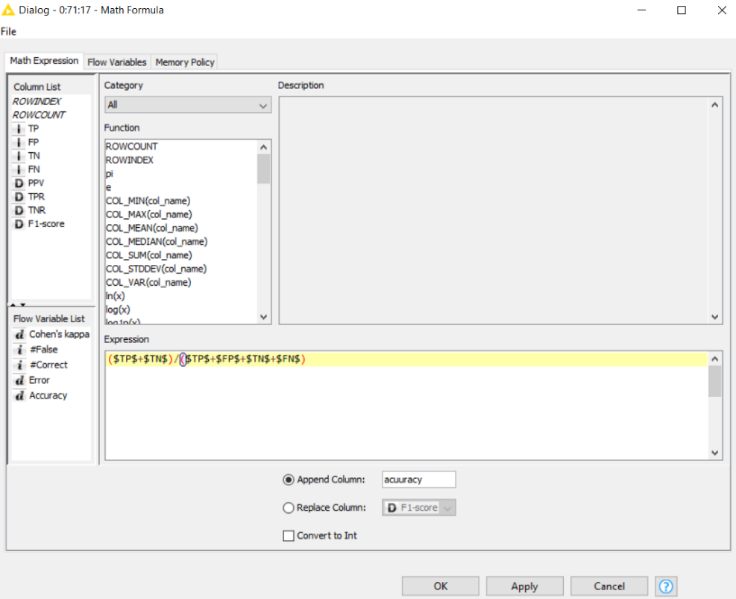


Las segundas salidas de todos los algoritmo, que contienen la matriz de confusión y las estadísticas de precisión, las vamos a concatenar y le vamos a añadir algunos datos más de precisión del algoritmo, con el nodo “Math Formula”: Accuraci,G\_mean.



Concatenación

Math Formula (Accuracy)



Math Formula (G\_mean)

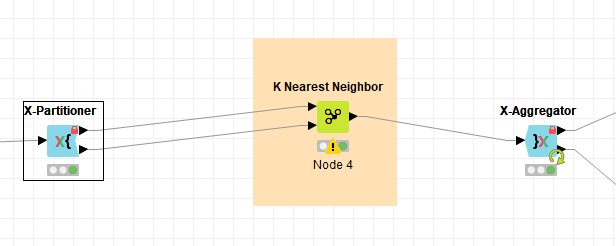
Por último, unimos el resultado del área bajo la curva roc y las estadísticas de precisión.

* **Configuración por defecto:**

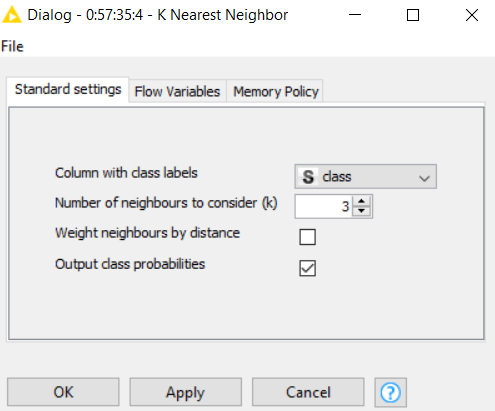
## 2.1.KNN

* Flujo de trabajo:

Validación cruzada con el algoritmo “KNN”.



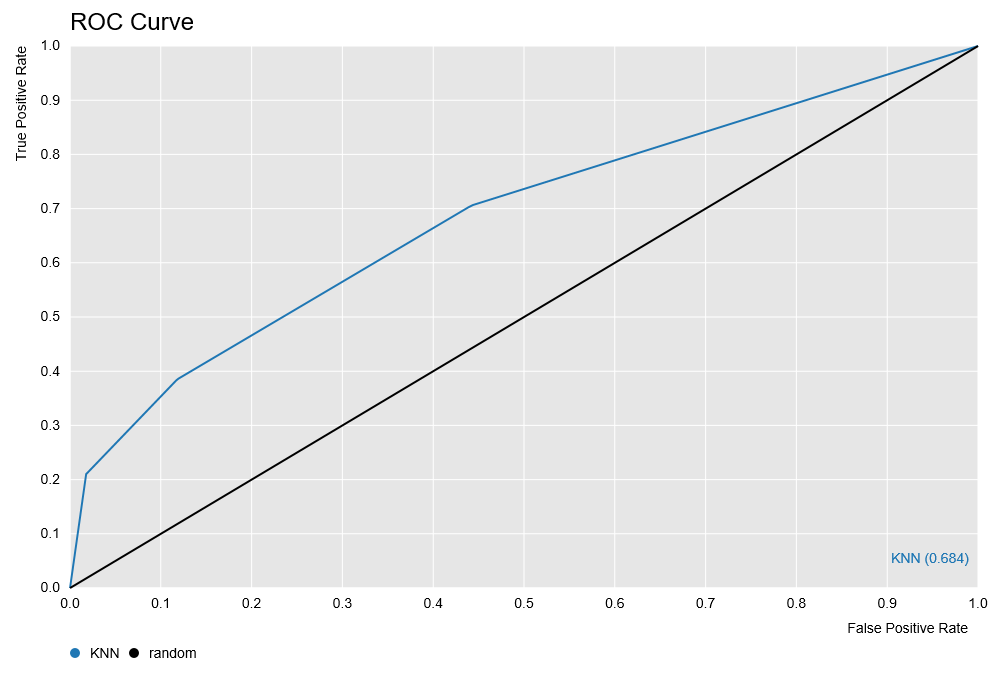
* Configuración del algoritmo “KNN”:



Configuración Knn por defecto

* Resultados del algoritmo “KNN”:

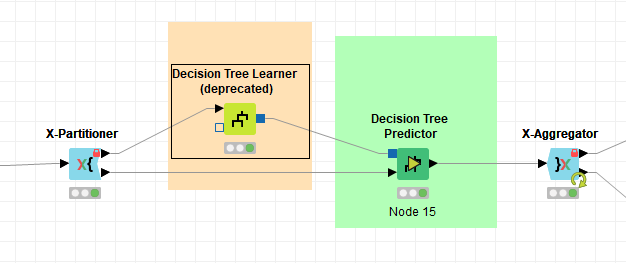
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| KNN | 4480 | 4352 | 32803 | 7207 | 0,507 | 0,383 | 0,993 | 0,437 | 0,763 | 0,582 | 0,684 |



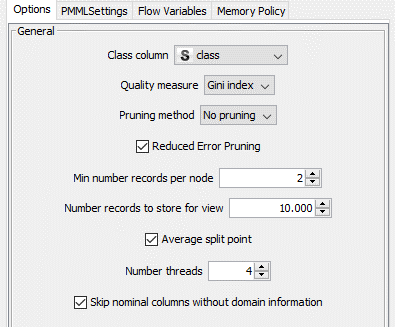
## 2.2.C4.5

* Flujo de trabajo:

Validación cruzada con el algoritmo “C4.5”.

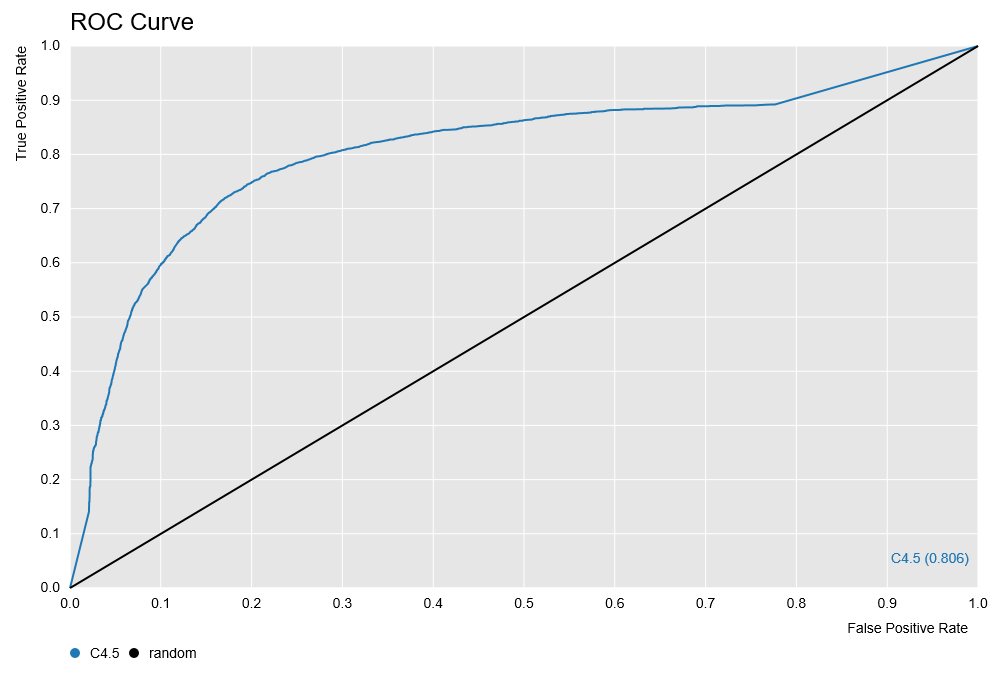


* Configuración del algoritmo “C4.5”:



* Resultados del algoritmo “C4.5”:

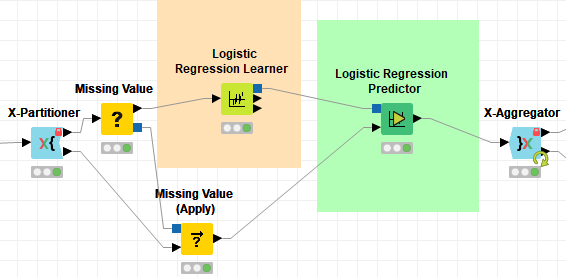
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| C4.5 | 6932 | 3647 | 33042 | 4550 | 0,655 | 0,604 | 0,901 | 0,628 | 0,830 | 0,737 | 0,806 |

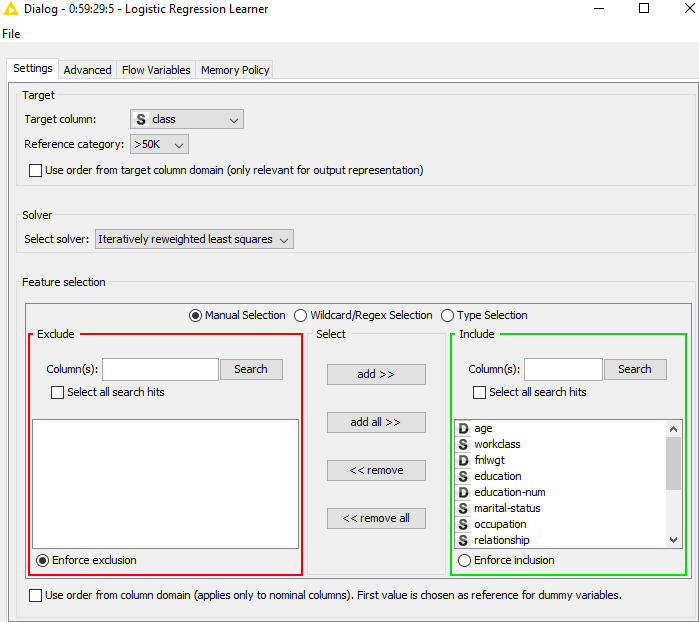


## 2.3. LOGISTIC REGRESSION

* Flujo de trabajo:

Validación cruzada con el algoritmo “Logistic Regression” y tratamiento de los valores perdidos.



* Configuración del algoritmo “Logistic Regression”: 
* Resultados del algoritmo “Logistic Regression”:

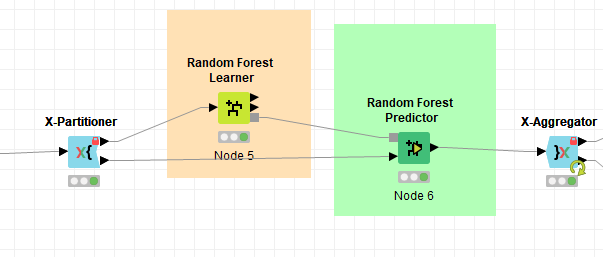
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| Logistic Regression | 6948 | 3505 | 34650 | 4739 | 0,735 | 0,595 | 0,933 | 0,657 | 0,852 | 0,745 | 0,904 |



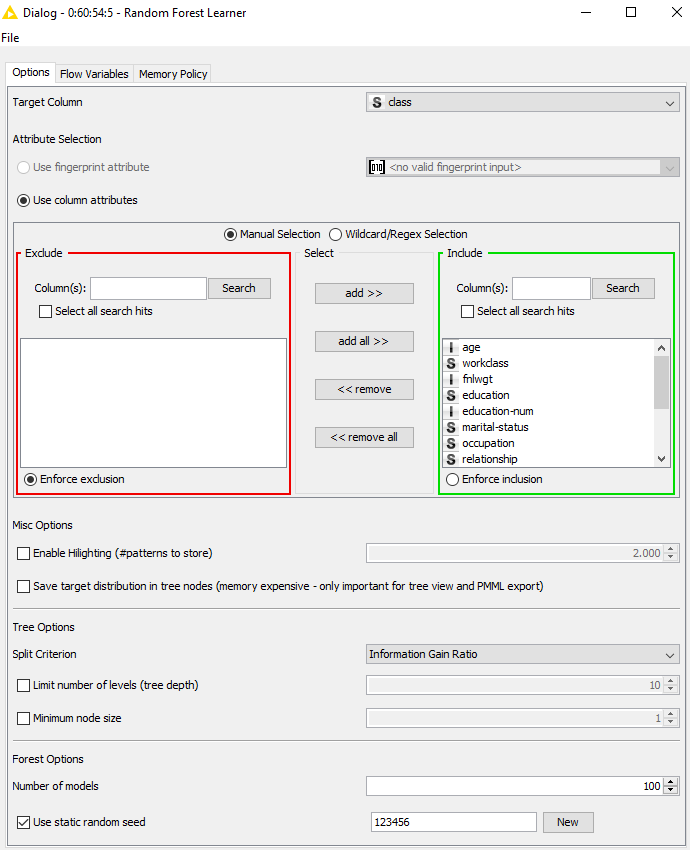
## 2.4. RANDOM FOREST

* Flujo de trabajo:

Validación cruzada con el algoritmo “Random Forest”.

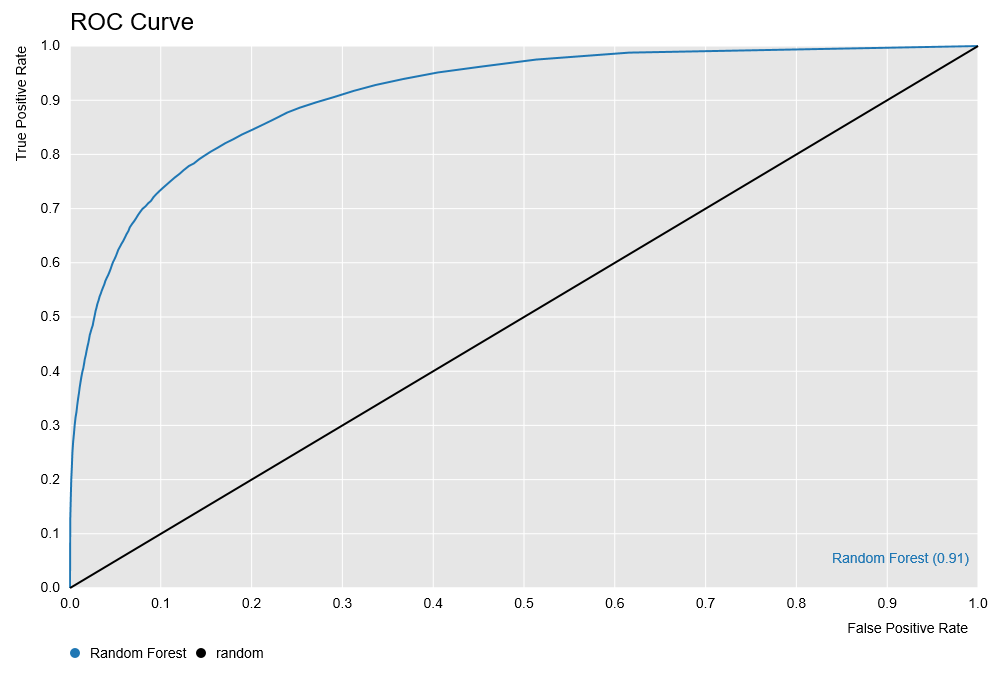


* Configuración del algoritmo “Random Forest”:



* Resultados del algoritmo “Random Forest”:

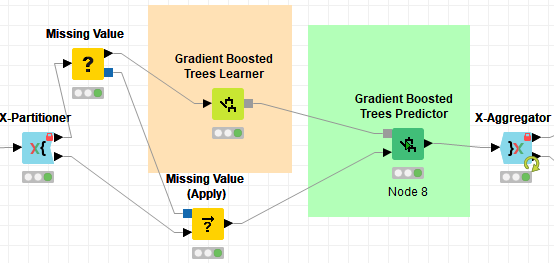
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| Random Forest | 7158 | 1877 | 35278 | 4529 | 0,792 | 0,612 | 0,949 | 0,691 | 0,869 | 0,763 | 0,910 |



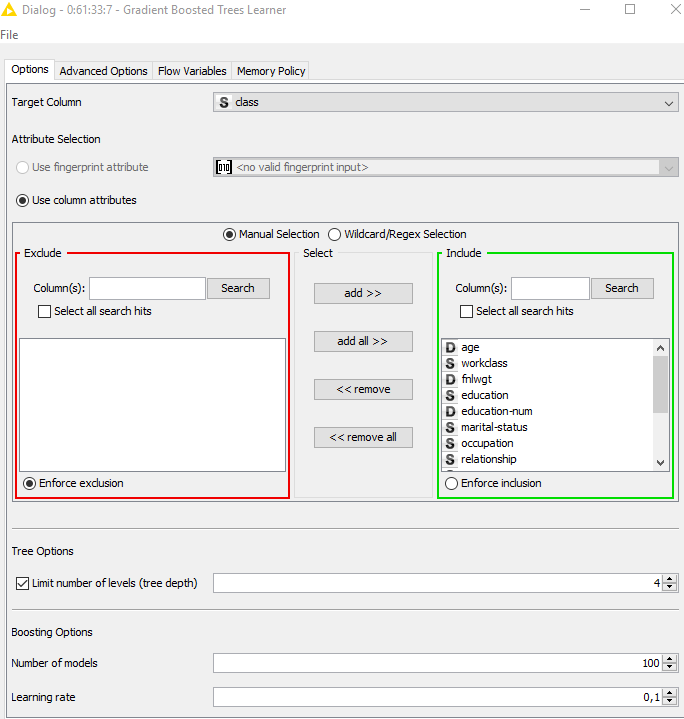
## 2.5. GRADIENT BOOSTED

* Flujo de trabajo:

Validación cruzada con el algoritmo “Gradient Boosted” y tratamiento de los valores perdidos.

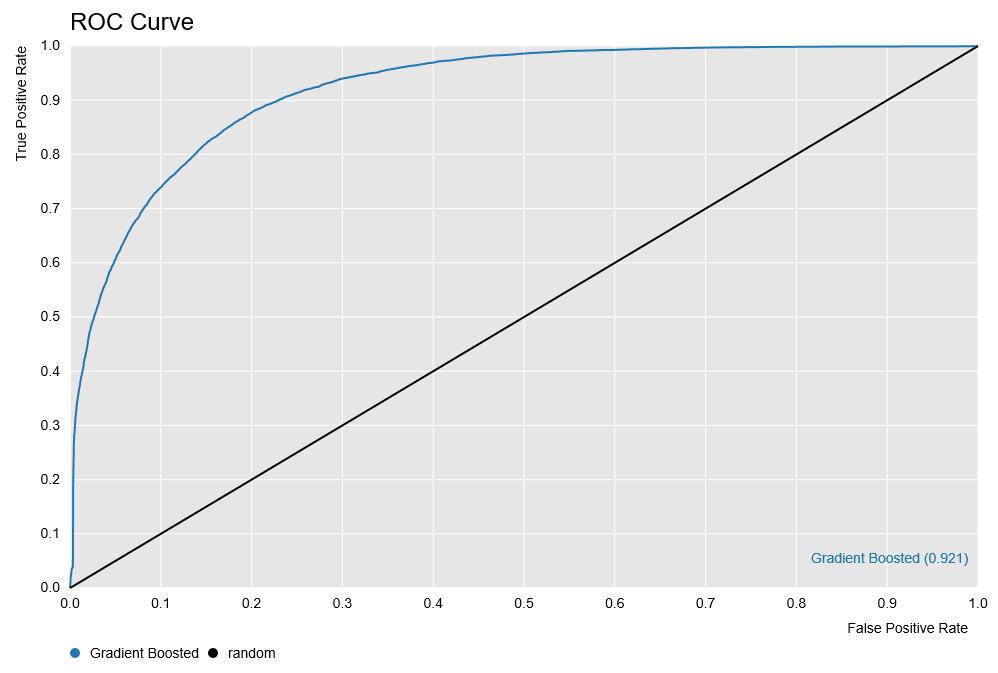


* Configuración del algoritmo “Gradient Boosted”:



* Resultados del algoritmo “Gradient Boosted”:

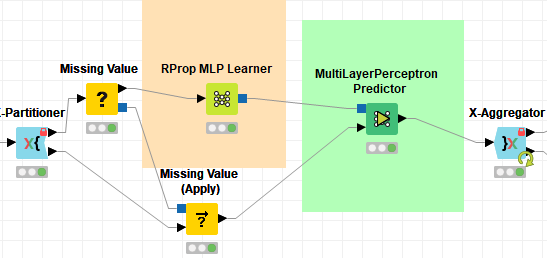
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| Gradient Boosted | 7370 | 2105 | 35050 | 4317 | 0,778 | 0,631 | 0,943 | 0,697 | 0,869 | 0,771 | 0,921 |



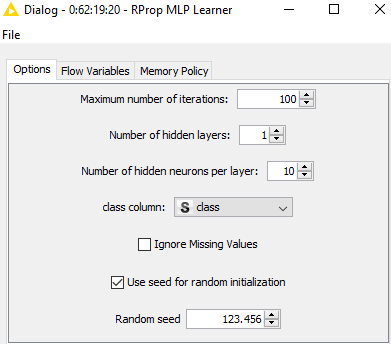
## 2.6. NETURAL NETWORKS

* Flujo de trabajo:

Validación cruzada con el algoritmo “Gradient Boosted” y tratamiento de los valores perdidos.

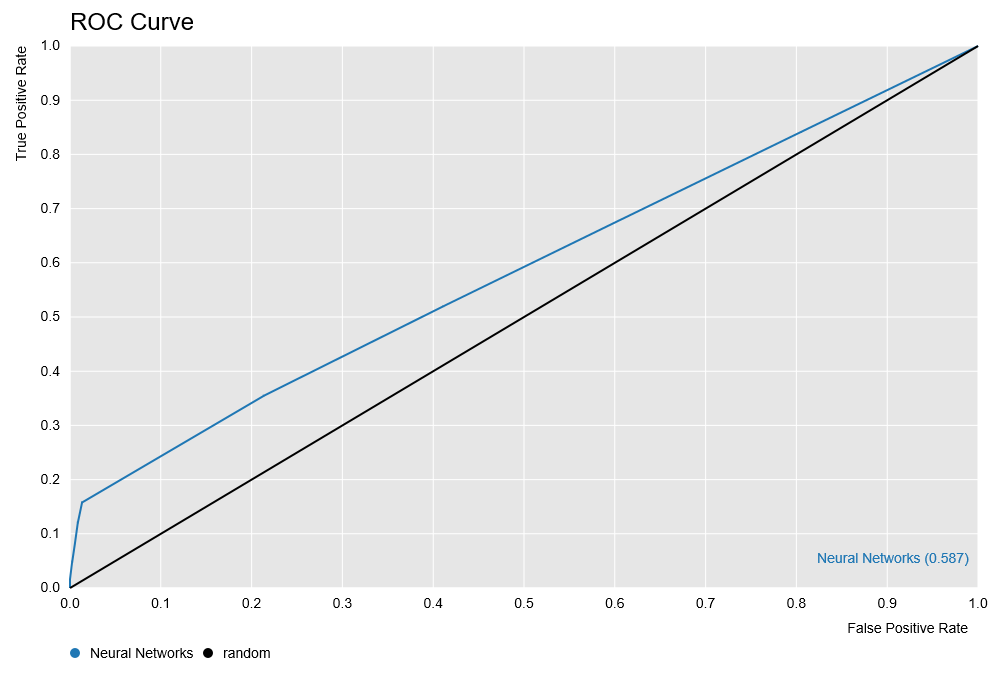


* Configuración del algoritmo “Neural Networks”:



* Resultados del algoritmo “Neural Networks”:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| Neural Networks | 1847 | 498 | 36657 | 9840 | 0,788 | 0,158 | 0,987 | 0,263 | 0,788 | 0,395 | 0,587 |



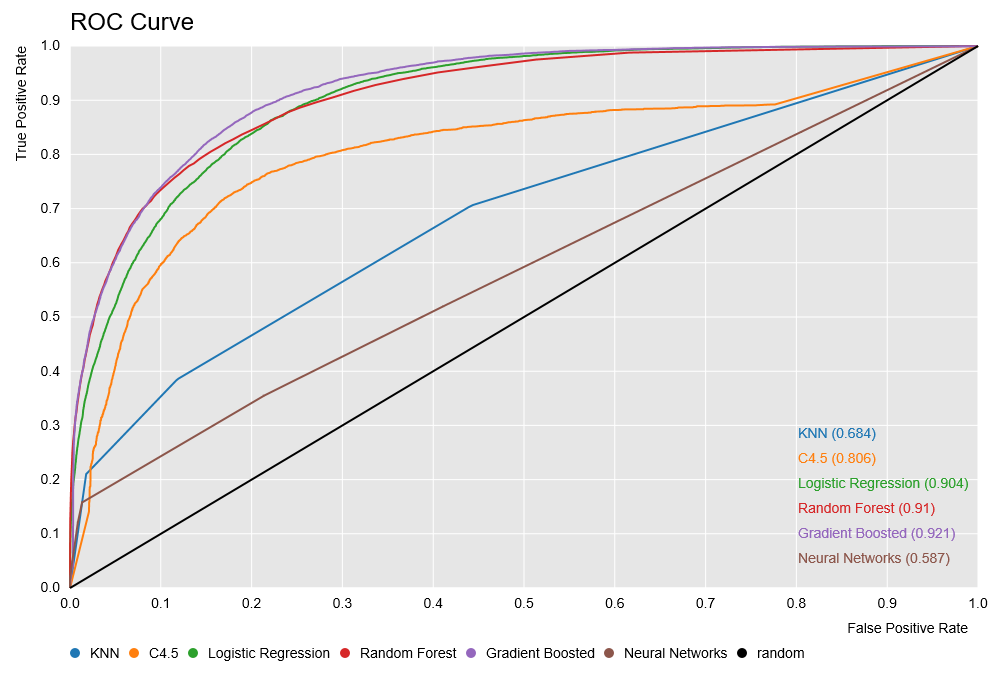
# 3.ANÁLISIS DE RESULTADOS

* **Resultados por defecto:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| KNN | 4480 | 4352 | 32803 | 7207 | 0,507 | 0,383 | 0,993 | 0,437 | 0,763 | 0,582 | 0,684 |
| C4.5 | 6932 | 3647 | 33042 | 4550 | 0,655 | 0,604 | 0,901 | 0,628 | 0,830 | 0,737 | 0,806 |
| Logistic Regression | 6948 | 3505 | 34650 | 4739 | 0,735 | 0,595 | 0,933 | 0,657 | 0,852 | 0,745 | 0,904 |
| Random Forest | 7158 | 1877 | 35278 | 4529 | 0,792 | 0,612 | 0,949 | 0,691 | 0,869 | 0,763 | 0,910 |
| Gradient Boosted | 7370 | 2105 | 35050 | 4317 | 0,778 | 0,631 | 0,943 | 0,697 | 0,869 | 0,771 | 0,921 |
| Neural Networks | 1847 | 498 | 36657 | 9840 | 0,788 | 0,158 | 0,987 | 0,263 | 0,788 | 0,395 | 0,587 |

En la tabla podemos observar los resultados de todos los algoritmos analizados, en los que mostramos distintas métricas en torno a los resultados de TP, TN, FP y FN.

* La columna **Accuracy**, es la precisión general del algoritmo, es decir, cuanto acierta en general, quizás la primera medida que se nos ocurre tomar. Los algoritmos “KNN” y “Neural Networks” son los que peores resultados obtienen seguramente debido a que necesitan más procesamiento previo de los datos, a continuación “C4.5” y el resto muy cercanos unos de otros, aunque en general no obtienen resultados muy dispares.
* En la columna **PPV**, tenemos el valor predictivo respecto a la clase positiva, es decir, cuanto de preciso es el algoritmo cuando predice la clase positiva, mientras que en **TPR** tenemos la tasa de acierto en torno a todos los valores de la clase positiva.
  + Hay algoritmos que se comportan muy mal tanto en **PPV** como en **TPR** como es “KNN”, otros como “Neural Networks” que, aunque pudiera parece que no es demasiado malo mirando en **PPV**, al observar **TPR** vemos como su porcentaje en **PPV** está basado en que la mayoría de predicciones las hace hacia la clase negativa (TN+ FN=46497 || TP + FP=2345).
  + En los otros 4 algoritmo, aunque hay unos mejores que otros, los 4 tienen en común que están relativamente equilibrados, siendo “Random Forest” y “Gradient Boosted” los mejores.
* En la columna **TNR**, tenemos la tasa de acierto en torno a todos los valores de la clase negativa. Aquí observamos que todos los algoritmos obtienen porcentajes más altos. Esto es debido a que la clase negativa es la clase mayoritaria, incluso vemos como “Neural Networks” obtiene un 98.7%, que como dijimos antes, es porque casi todas sus predicciones son hacia la clase negativa.
* La columna **F1\_score** nos da una medida que combina **PPV** y **TPR**. Aquí se ve reflejado los comentado anteriormente, que el algoritmo “KNN” y “Neural Networks”, obtienen malos resultados y que los otros 4 algoritmos están más equilibrados siendo los mejores “Random Forest” y “Gradient Boosted”.
* La columna **G\_mean**, nos da la media geométrica, que no es tan sensible a los valores extremos como la media y de nuevos vemos resultados que verifican lo comentado anteriormente, aunque podemos apreciar como “Neural Networks” se ve más penalizado en este caso que “KNN” por estar más desbalanceada hacia la clase negativa.
* **ROC**: nos muestra una predicción continua de **TPR** frente a **FPR** (1-TNR). De nuevo, los algoritmos que están más equilibrados obtienen mejores resultados y los menos equilibrados peores resultados, siendo el peor “Neural Networks” y el mejor “Gradient Boosted”.

****

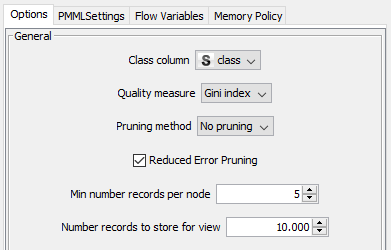
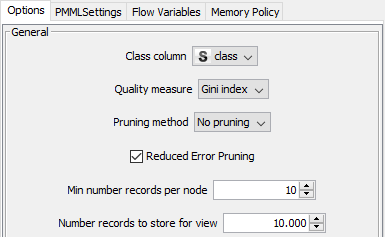
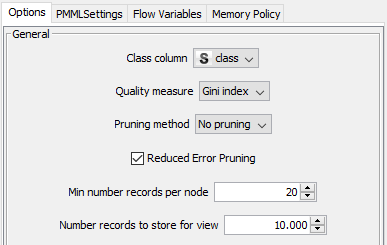
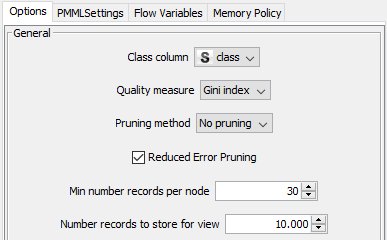
* Conclusión:
  + No siempre lo más importante es la precisión del algoritmo (Accuracy) y se debe tener en cuenta lo equilibrado que está el algoritmo en general.
  + Hay algoritmo que por defecto funcionan relativamente bien y otros que requieren muchas configuraciones para obtener unos buenos resultados, por ejemplo “Neural Networks”, un algoritmo muy potente, pero que requiere de muchas pruebas de experimentación y cambios en la configuración para obtener buenos resultados.
  + Hay algoritmos que le influye más el tratamiento inicial de los datos que a otros. Por ejemplo, para los algoritmos “KNN” y “Neural Networks” cuando los valores no están normalizados, no obtienen buenos resultados, aunque al resto de algoritmos también pueda afectarles.

# 4.CONFIGURACIÓN DE ALGORITMOS

Como base para la comparación con las distintas configuraciones vamos a usar el resultado obtenido tras el procesado de datos en el punto 5.

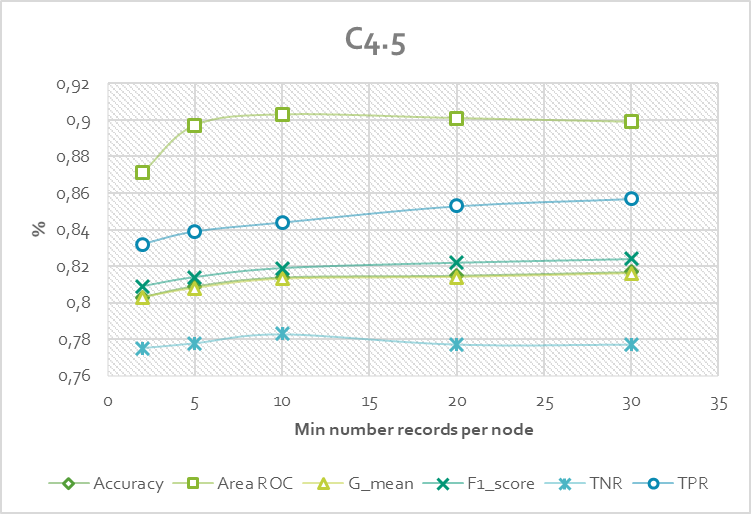
## 4.1. DECISION TREE (C4.5)

* **Número de registros mínimos por nodo**: vamos a aumentarlos teniendo en cuenta dos aspectos: por un lado, para el caso de que hubiera sobreajuste en el aprendizaje y por otro lado para el caso de que mejore, también sería un modelo más sencillo de explicar.

### RESULTADOS

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | Records per node | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| C4.5 | 2 | 9718 | 2630 | 9057 | 1969 | 0,787 | 0,832 | 0,775 | 0,809 | 0,803 | 0,803 | 0,871 |
| C4.5 | 5 | 9805 | 2594 | 9093 | 1882 | 0,791 | 0,839 | 0,7778 | 0,814 | 0,809 | 0,808 | 0,897 |
| C4.5 | 10 | 9868 | 2536 | 9151 | 1819 | 0,796 | 0,844 | 0,783 | 0,819 | 0,814 | 0,813 | 0,903 |
| C4.5 | 20 | 9970 | 2603 | 9084 | 1717 | 0,793 | 0,853 | 0,777 | 0,822 | 0,815 | 0,814 | 0,901 |
| C4.5 | 30 | 10015 | 2601 | 9089 | 1672 | 0,794 | 0,857 | 0,777 | 0,824 | 0,817 | 0,816 | 0,899 |



#### ANÁLISIS

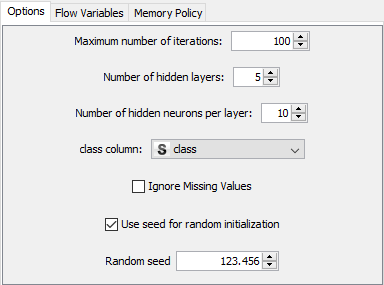
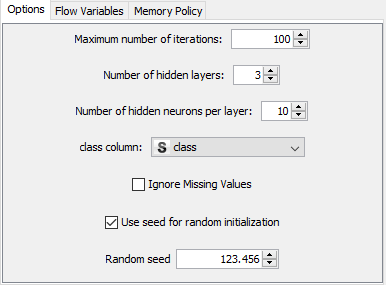
Observamos que claramente el algoritmo estaba sobreajustando en los datos de train, ya que al aumentar el número mínimo de instancias por nodo hoja del árbol, hacemos una poda y evitamos que el árbol se especialice en exceso para cada caso. En los 4 experimento que hemos hecho, mejoran los resultados, aunque a partir de 10 instancias por hoja, obtiene peores resultados tanto en ROC como TNR y apenas mejora en el resto, por lo que nuestra mejor opción sería 10 o 20 instancias por hoja, dependiendo de lo que fuera más importante en nuestro problema. Además de mejorar en los resultados, obtenemos un árbol más sencillo que sería más fácil de explicar.

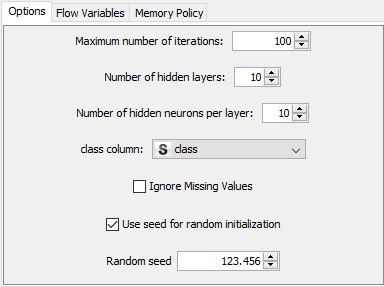
## 4.2. NEURAL NETWORKS

Las redes neuronales tienen muchos posibles parámetros que podemos modificar, pero vamos a hacer algunos cambios sólo con respecto a la estructura de la red.

* **Número de capas ocultas:** Al aumentar el número de capas ocultas, estamos dando la posibilidad a la red de ajustar más el aprendizaje, aunque si el número de capas va creciendo y no tenemos un número suficiente de ejemplos en nuestro conjunto de train, empezaremos a sobreajustar y perderemos precisión en test. También hay que tener en cuenta que cuantas más capas usemos, más tiempo tardará el aprendizaje.

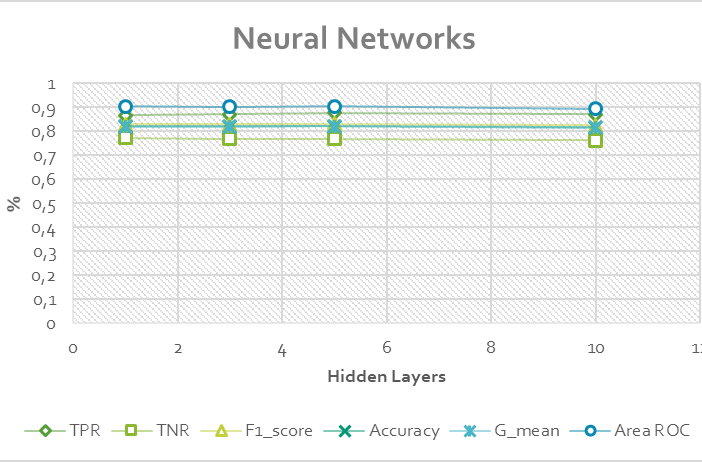
En el ejemplo base teníamos 1 capa oculta:





### RESULTADOS

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | hidden layers | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| Neural Net | 1 | 10147 | 2655 | 9032 | 1540 | 0,793 | 0,868 | 0,773 | 0,829 | 0,821 | 0,819 | 0,905 |
| Neural Net | 3 | 10191 | 2699 | 8988 | 1496 | 0,791 | 0,872 | 0,769 | 0,829 | 0,821 | 0,8189 | 0,903 |
| Neural Net | 5 | 10246 | 2708 | 8979 | 1441 | 0,791 | 0,877 | 0,768 | 0,832 | 0,822 | 0,821 | 0,905 |
| Neural Net | 10 | 10175 | 2781 | 8906 | 1512 | 0,785 | 0,871 | 0,762 | 0,826 | 0,816 | 0,815 | 0,894 |

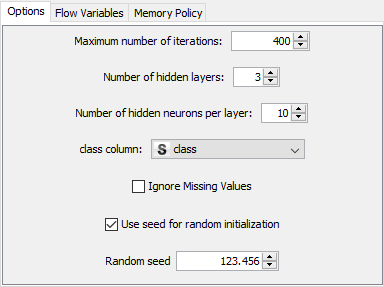
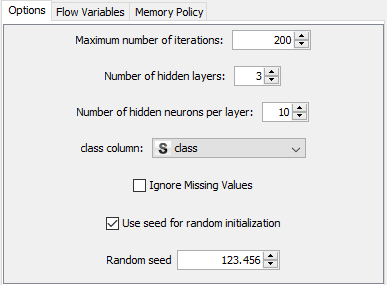


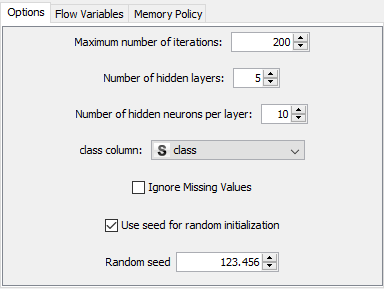
#### ANÁLISIS

Podemos ver como con 3 capas ocultas apenas han cambiado los resultados. Con 5 capas ocultas, hemos mejorado algo, aunque no demasiado y con 10 capas ocultas hemos perdido algo de calidad en el aprendizaje.

Esto puede deberse, en el último caso, quizás a sobreaprendizaje, pero también puede deberse a que el número de iteraciones máximas no sea suficiente para que el algoritmo converja, por eso, vamos a tomar 3 y 5 capas ocultas y vamos a cambiar el número de iteraciones máximas.

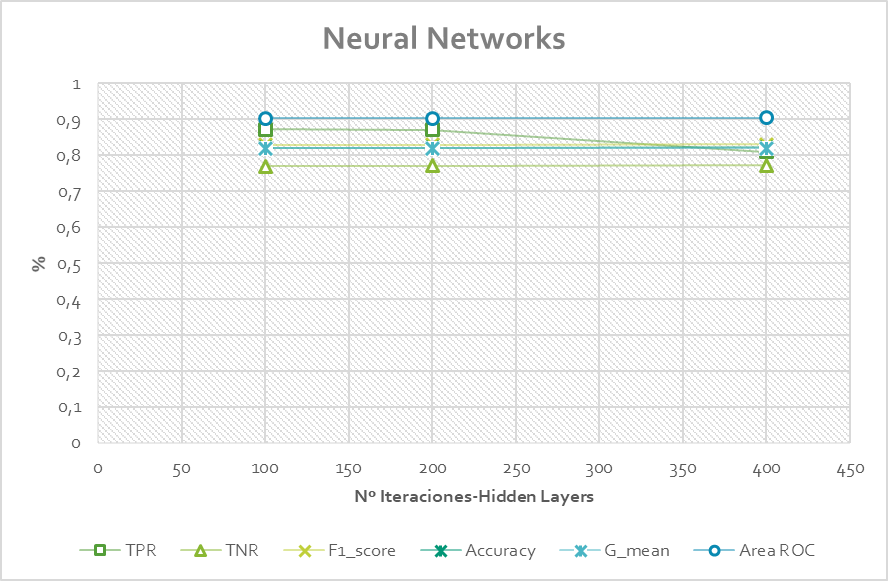
* **Número de iteraciones máximas:** vamos a ir subiendo el número de iteraciones máximas para ver si mejoramos los resultados anteriores.





### RESULTADOS

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | Nº iter | hidden layers | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| Neural Net | 100 | 1 | 10191 | 2699 | 8988 | 1496 | 0,791 | 0,872 | 0,769 | 0,829 | 0,821 | 0,819 | 0,903 |
| Neural Net | 200 | 3 | 10180 | 2678 | 9009 | 1507 | 0,792 | 0,871 | 0,771 | 0,829 | 0,821 | 0,819 | 0,903 |
| Neural Net | 400 | 5 | 10178 | 2660 | 9027 | 1509 | 0,793 | 0,81 | 0,772 | 0,83 | 0,822 | 0,82 | 0,904 |



#### ANÁLISIS

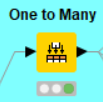
Hemos probado distintas combinaciones entre capas ocultas e iteraciones y no hemos conseguido mejorar resultados. Una posibilidad sería que el número de datos (teniendo en cuenta que hemos equilibrado las clases y perdido muchos datos) es muy pequeño para que la red neuronal pueda tener un número de capas suficiente como para tener mejores resultados. También es posible que el “learning rate” que tenga por defecto la red sea demasiado grande o demasiado pequeño.

# 5.PROCESADO DE DATOS

## 5.1.ONE TO MANY

El primer procesado que vamos a realizar es convertir los datos categóricos. Podríamos optar por convertir las variables “string” a “numéricas”, pero creo que es mejor opción crear una nueva columna por cada categoría en la que si es esa categoría, esté a 1 y si no, esté a 0.

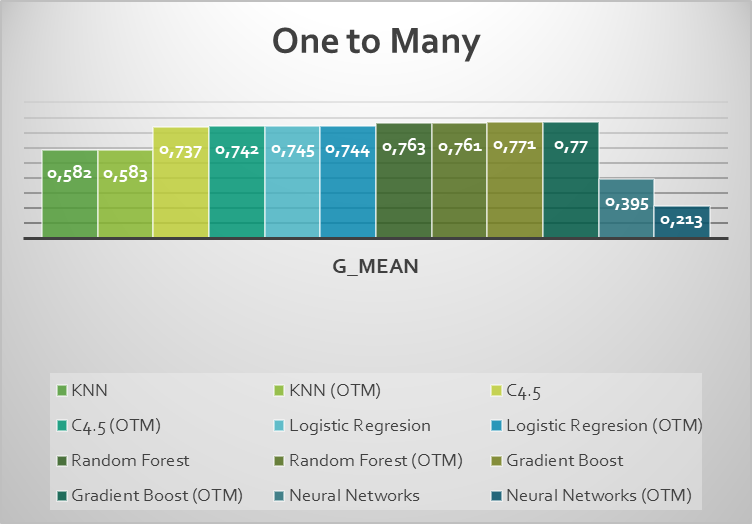
Para esto, usamos el nodo “one to many”:



### RESULTADOS

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| KNN | 4480 | 4352 | 32803 | 7207 | 0,507 | 0,383 | 0,883 | 0,437 | 0,763 | 0,582 | 0,684 |
| KNN (OTM) | 4492 | 4272 | 32883 | 7195 | 0,513 | 0,384 | 0,885 | 0,439 | 0,765 | 0,583 | 0,687 |
| C4.5 | 6932 | 3647 | 33042 | 4550 | 0,655 | 0,604 | 0,901 | 0,628 | 0,83 | 0,737 | 0,806 |
| C4.5 (OTM) | 7219 | 4052 | 33103 | 4468 | 0,64 | 0,618 | 0,891 | 0,629 | 0,826 | 0,742 | 0,783 |
| Logistic Regression | 6948 | 2505 | 34650 | 4739 | 0,735 | 0,595 | 0,933 | 0,657 | 0,852 | 0,745 | 0,904 |
| Logistic Regression (OTM) | 6933 | 2532 | 34623 | 4754 | 0,732 | 0,593 | 0,932 | 0,656 | 0,851 | 0,744 | 0,905 |
| Random Forest | 7158 | 1877 | 35278 | 4529 | 0,792 | 0,612 | 0,949 | 0,691 | 0,869 | 0,763 | 0,91 |
| Random Forest (OTM) | 7136 | 1904 | 35251 | 4551 | 0,789 | 0,611 | 0,949 | 0,689 | 0,868 | 0,761 | 0,893 |
| Gradient Boost | 7370 | 2105 | 35050 | 4317 | 0,778 | 0,631 | 0,943 | 0,697 | 0,869 | 0,771 | 0,921 |
| Gradient Boost (OTM) | 7324 | 1965 | 35190 | 4363 | 0,788 | 0,6272 | 0,947 | 0,698 | 0,87 | 0,77 | 0,925 |
| Neural Networks | 1847 | 498 | 36657 | 9840 | 0,788 | 0,158 | 0,987 | 0,263 | 0,788 | 0,395 | 0,587 |
| Neural Networks (OTM) | 533 | 28 | 37127 | 11154 | 0,95 | 0,046 | 0,999 | 0,087 | 0,771 | 0,213 | 0,528 |





#### ANÁLISIS

Mostramos las gráficas de Accuracy y G\_mean como ejemplo, aunque en todas las medidas se comporta de forma similar.

En general vemos que prácticamente no mejora ningún algoritmo e incluso alguno empeora algo.

Creo que esto puede ser debido a que haya variables que no aporten mucho o nada al aprendizaje y esto sumado a que “one to many” crea muchas columnas nuevas, puede hacer que empeore.

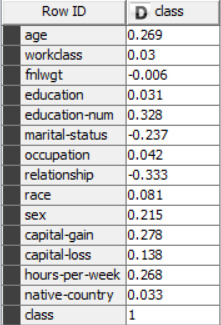
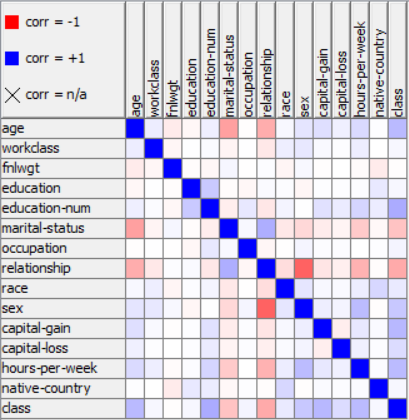
Para tratar de solucionar esto, el siguiente cambio que voy a hacer, será comprobar la correlación entre la clase y las distintas variables para tratar de eliminar alguna.

## 5.3. CORRELACIÓN

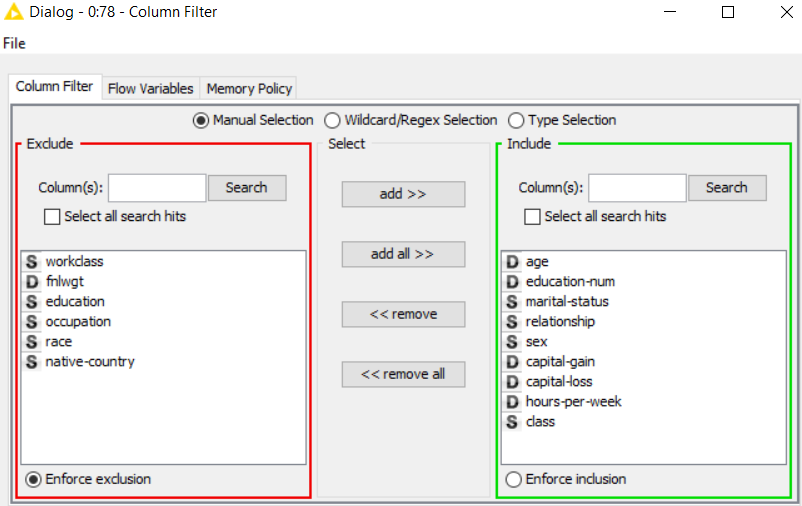
Para tratar de eliminar variables, vamos a estudiar la correlación entre la clase y las distintas variables con el modulo “Rank Correlation”. Dado que no debemos utilizar los datos de test para ver la correlación entre las variables y que hasta el momento de hacer validación cruzada no sabemos cuáles serán train y cuales test, utilizamos X-Partitioner para obtener la misma partición en train y test que se va a utilizar en cada algoritmo para el aprendizaje en la correlación cruzada. A continuación, tratamos los valores perdidos para que “Rank Correlation” funcione correctamente y por último obtenemos la correlación.

Nos vamos a fijar en los valores de correlación de cada variable con respecto a la clase.





Vamos a tomar como medida el que las variables que tengan un valor inferior a 0,1 en valor absoluto, la eliminamos. Este es un criterio propio, que considera que valores por debajo de 0.1 no van a tener mucha influencia. Mediante experimentación podríamos encontrar el valor más adecuado.



### RESULTADOS

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| KNN | 4480 | 4352 | 32803 | 7207 | 0,507 | 0,383 | 0,883 | 0,437 | 0,763 | 0,582 | 0,684 |
| KNN (RC) | 6931 | 2430 | 34725 | 4756 | 0,84 | 0,593 | 0,935 | 0,659 | 0,853 | 0,744 | 0,885 |
| C4.5 | 6932 | 3647 | 33042 | 4550 | 0,655 | 0,604 | 0,901 | 0,628 | 0,83 | 0,737 | 0,806 |
| C4.5 (RC) | 7104 | 3086 | 34069 | 4583 | 0,697 | 0,608 | 0,9017 | 0,649 | 0,843 | 0,747 | 0,877 |
| Logistic Regression | 6948 | 2505 | 34650 | 4739 | 0,735 | 0,595 | 0,933 | 0,657 | 0,852 | 0,745 | 0,904 |
| Logistic Regression (RC) | 6678 | 2630 | 34525 | 5009 | 0,717 | 0,571 | 0,929 | 0,636 | 0,844 | 0,729 | 0,897 |
| Random Forest | 7158 | 1877 | 35278 | 4529 | 0,792 | 0,612 | 0,949 | 0,691 | 0,869 | 0,763 | 0,91 |
| Random Forest (RC) | 6266 | 1498 | 35657 | 5421 | 0,807 | 0,536 | 0,96 | 0,644 | 0,858 | 0,717 | 0,869 |
| Gradient Boost | 7370 | 2105 | 35050 | 4317 | 0,778 | 0,631 | 0,943 | 0,697 | 0,869 | 0,771 | 0,921 |
| Gradient Boost (RC) | 7046 | 1971 | 35184 | 4641 | 0,781 | 0,603 | 0,947 | 0,681 | 0,865 | 0,756 | 0,921 |
| Neural Networks | 1847 | 498 | 36657 | 9840 | 0,788 | 0,158 | 0,987 | 0,263 | 0,788 | 0,395 | 0,587 |
| Neural Networks (RC) | 5447 | 2456 | 34697 | 6540 | 0,689 | 0,466 | 0,934 | 0,556 | 0,822 | 0,66 | 0,84 |

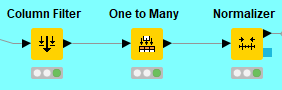
#### ANÁLISIS

* En este caso, sí que vemos una mejora importante en los algoritmos “KNN” y “Neural Networks”. “C4.5”, también mejora ligeramente, mientras que el resto de algoritmos, apenas cambian. Con esto vemos que hay algoritmos a los que les afecta muchos más que a otros el tratamiento de los datos. “Random forest” y “Logistic Regression”, incluso llegan a ser ligeramente peores con estos cambios, algo que puede ser debido a que, a estos algoritmos, el aumento de columnas(variables) con “one to many”, les perjudica y funcionan mejor cuando hay pocas variables. “Gradient Boosted” apenas cambia, aunque se observa que empeora ligeramente en TPR y mejora en TNR, por lo que quizás también le afecte algo el aumento de variables.
* Las mejoras en los algoritmos “KNN” y “Neural Networks”, son varias, por un lado, la precisión del algoritmo (**accuracy)**, que, aunque ya comentamos que no siempre es la mejor medida que podemos hacer, si es muy importante. En la columna **TNR** podemos observar para “Neural Networks”, como ha empeorado ligeramente, pero esto en este caso es algo positivo, porque es un indicativo junto al aumento considerable en **TPR** de que ya no está tan desbalanceada hacia la clase negativa. En el caso de “KNN” la mejora ha sido en ambos casos, es mejor en la clase positiva y negativa, y por tanto más equilibrada.
* Por último, en la curva **ROC**, se observa todo esto que he comentado, con el aumento del área bajo la curva en “KNN”,” Neural Networks” y “C4.5”, una ligera pérdida en “Logistic Regression” y “Random Forest”, mientras “Gradient Boosted” se mantiene igual.

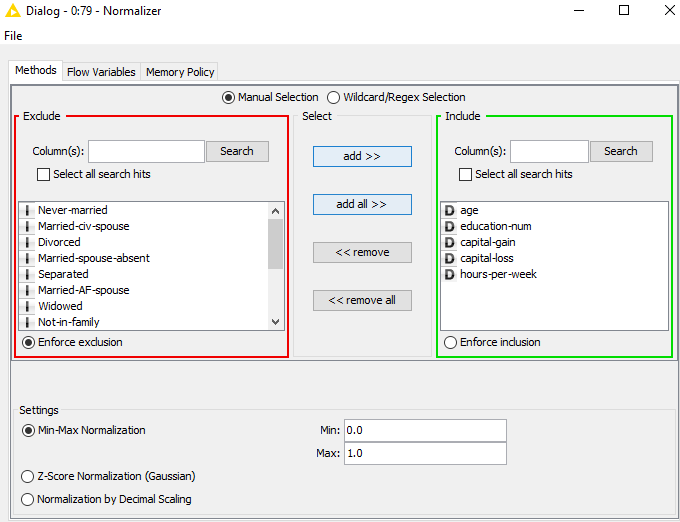
## 5.4. NORMALIZAR

Otro aspecto importante para las variables reales es el hecho de que estén normalizadas. Si lo datos no están normalizados, los valores muy grandes o muy pequeños pueden tener mucha más influencia que el resto sobre el aprendizaje.

Para esto vamos a utilizar el nodo ”Normalizer**”.**



Vamos a hacer 3 tipos de normalización(Settings) y compararemos los resultados obtenidos en cada uno**:**

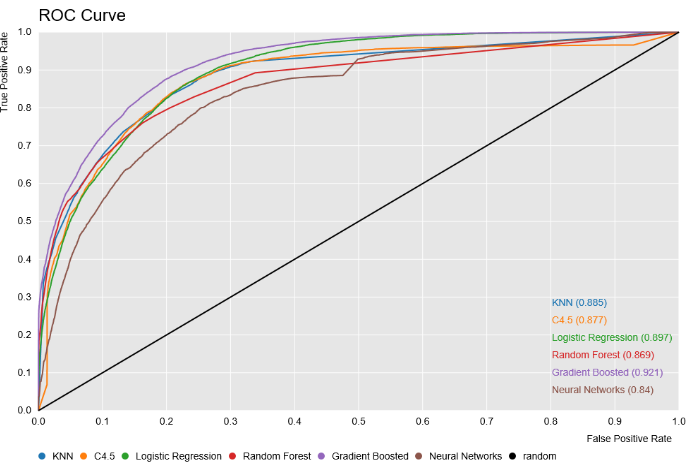


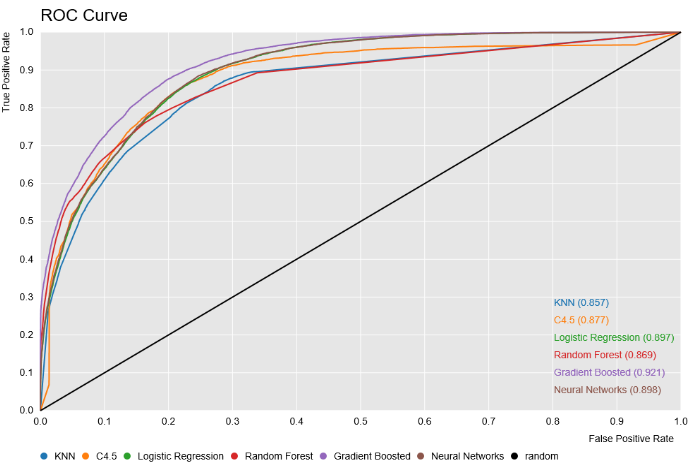
### RESULTADOS

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| KNN (RC) | 6931 | 2430 | 34725 | 4756 | 0,84 | 0,593 | 0,935 | 0,659 | 0,853 | 0,744 | 0,885 |
| KNN (NOR1) | 6317 | 2774 | 34381 | 5370 | 0,695 | 0,541 | 0,925 | 0,608 | 0,833 | 0,707 | 0,855 |
| C4.5 (RC) | 7104 | 3086 | 34069 | 4583 | 0,697 | 0,608 | 0,9017 | 0,649 | 0,843 | 0,747 | 0,877 |
| C4.5 (NOR1) | 7103 | 3090 | 34065 | 4584 | 0,697 | 0,608 | 0,917 | 0,649 | 0,843 | 0,746 | 0,877 |
| Logistic Regression (RC) | 6678 | 2630 | 34525 | 5009 | 0,717 | 0,571 | 0,929 | 0,636 | 0,844 | 0,729 | 0,897 |
| Logistic Regression (NOR1) | 6678 | 2630 | 34525 | 5009 | 0,717 | 0,571 | 0,929 | 0,636 | 0,844 | 0,729 | 0,897 |
| Random Forest (RC) | 6265 | 1498 | 35657 | 5421 | 0,807 | 0,536 | 0,96 | 0,644 | 0,858 | 0,717 | 0,869 |
| Random Forest (NOR1) | 6265 | 1498 | 35657 | 5422 | 0,807 | 0,536 | 0,96 | 0,644 | 0,858 | 0,717 | 0,869 |
| Gradient Boost (RC) | 7046 | 1971 | 35184 | 4641 | 0,781 | 0,603 | 0,947 | 0,681 | 0,865 | 0,756 | 0,921 |
| Gradient Boost (NOR1) | 7045 | 1971 | 35184 | 4642 | 0,781 | 0,603 | 0,974 | 0,681 | 0,865 | 0,756 | 0,921 |
| Neural Networks (RC) | 5447 | 2456 | 34697 | 6540 | 0,689 | 0,466 | 0,934 | 0,556 | 0,822 | 0,66 | 0,84 |
| Neural Networks (NOR1) | 6746 | 2604 | 34551 | 4941 | 0,721 | 0,577 | 0,93 | 0,641 | 0,846 | 0,733 | 0,902 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| KNN (RC) | 6931 | 2430 | 34725 | 4756 | 0,84 | 0,593 | 0,935 | 0,659 | 0,853 | 0,744 | 0,885 |
| KNN (NOR2) | 6554 | 2722 | 34433 | 5133 | 0,707 | 0,561 | 0,927 | 0,625 | 0,839 | 0,721 | 0,862 |
| C4.5 (RC) | 7104 | 3086 | 34069 | 4583 | 0,697 | 0,608 | 0,917 | 0,649 | 0,843 | 0,747 | 0,877 |
| C4.5 (NOR2) | 7106 | 3088 | 34067 | 4581 | 0,697 | 0,608 | 0,917 | 0,65 | 0,843 | 0,747 | 0,877 |
| Logistic Regression (RC) | 6678 | 2630 | 34525 | 5009 | 0,717 | 0,571 | 0,929 | 0,636 | 0,844 | 0,729 | 0,897 |
| Logistic Regression (NOR2) | 6678 | 2630 | 34525 | 5009 | 0,717 | 0,571 | 0,929 | 0,636 | 0,844 | 0,729 | 0,897 |
| Random Forest (RC) | 6265 | 1498 | 35657 | 5421 | 0,807 | 0,536 | 0,96 | 0,644 | 0,858 | 0,717 | 0,869 |
| Random Forest (NOR2) | 6266 | 1497 | 35658 | 5421 | 0,807 | 0,536 | 0,96 | 0,644 | 0,858 | 0,717 | 0,869 |
| Gradient Boost (RC) | 7046 | 1971 | 35184 | 4641 | 0,781 | 0,603 | 0,947 | 0,681 | 0,865 | 0,756 | 0,921 |
| Gradient Boost (NOR2) | 7045 | 1970 | 35185 | 4642 | 0,781 | 0,603 | 0,94 | 0,681 | 0,865 | 0,756 | 0,921 |
| Neural Networks (RC) | 5447 | 2456 | 34697 | 6540 | 0,689 | 0,466 | 0,934 | 0,556 | 0,822 | 0,66 | 0,84 |
| Neural Networks (NOR2) | 6757 | 2290 | 34865 | 4930 | 0,747 | 0,578 | 0,938 | 0,652 | 0,852 | 0,737 | 0,908 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| KNN (RC) | 6931 | 2430 | 34725 | 4756 | 0,84 | 0,593 | 0,935 | 0,659 | 0,853 | 0,744 | 0,885 |
| KNN (NOR3) | 6370 | 2738 | 34417 | 5317 | 0,699 | 0,545 | 0,926 | 0,613 | 0,835 | 0,711 | 0,857 |
| C4.5 (RC) | 7104 | 3086 | 34069 | 4583 | 0,697 | 0,608 | 0,917 | 0,649 | 0,843 | 0,747 | 0,877 |
| C4.5 (NOR3) | 7108 | 3088 | 34067 | 4579 | 0,697 | 0,608 | 0,917 | 0,65 | 0,843 | 0,747 | 0,877 |
| Logistic Regression (RC) | 6678 | 2630 | 34525 | 5009 | 0,717 | 0,571 | 0,929 | 0,636 | 0,844 | 0,729 | 0,897 |
| Logistic Regression (NOR3) | 6679 | 2630 | 34525 | 5009 | 0,717 | 0,571 | 0,929 | 0,636 | 0,844 | 0,729 | 0,897 |
| Random Forest (RC) | 6265 | 1498 | 35657 | 5421 | 0,807 | 0,536 | 0,96 | 0,644 | 0,858 | 0,717 | 0,869 |
| Random Forest (NOR3) | 6266 | 1497 | 35658 | 5421 | 0,807 | 0,536 | 0,96 | 0,644 | 0,858 | 0,717 | 0,869 |
| Gradient Boost (RC) | 7046 | 1971 | 35184 | 4641 | 0,781 | 0,603 | 0,947 | 0,681 | 0,865 | 0,756 | 0,921 |
| Gradient Boost (NOR3) | 7046 | 1970 | 35185 | 4641 | 0,781 | 0,603 | 0,947 | 0,681 | 0,865 | 0,756 | 0,921 |
| Neural Networks (RC) | 5447 | 2456 | 34697 | 6540 | 0,689 | 0,466 | 0,934 | 0,556 | 0,822 | 0,66 | 0,84 |
| Neural Networks (NOR3) | 6826 | 2776 | 34379 | 4861 | 0,711 | 0,584 | 0,925 | 0,641 | 0,844 | 0,735 | 0,898 |

****

****

ROC: Antes-Después Normalización

#### ANÁLISIS

Aquí vemos como sólo dos algoritmos son sensibles a la normalización de los datos, “KNN” y “Neural Networks”, el resto obtienen exactamente los mismos resultados.

El algoritmo “KNN” empeora ligeramente en todos los casos, y quizás la explicación pueda ser que esté sobreajustando en los datos de train.

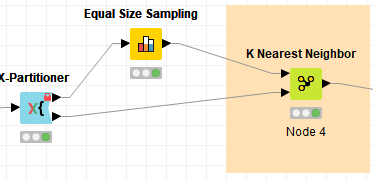
El algoritmo “Neural Networks” sí que obtiene mejores resultados al normalizar y empieza a posicionarse ya como uno de los mejores algoritmos en los experimentos.

La normalización que mejores resultados ha dado ha sido z-score (Normalizar 2) como podemos observar en esta última gráfica, por tanto, será la que vamos a utilizar en el resto de experimentos.

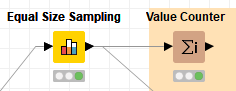
## 5.4. EQUILIBRAR CLASES

Estamos viendo como la clase mayoritaria influye mucho en el aprendizaje de los algoritmos, de forma que TNR tiene unos porcentajes muy altos, mientras que en TPR tiene porcentajes muy pequeños. Por ello, habría que equilibrar las dos clases y la forma más sencilla, aunque quizás no sea la mejor, es eliminar elementos de la clase mayoritaria de forma aleatoria hasta que las clases estén equilibradas.

Debemos hacerlo sólo sobre los datos de “train”, por lo tanto, hay que hacerlo una vez se han particionado los datos en el nodo “cross validation”.

Esto lo vamos a hacer con el nodo “Equal Size Sampling”. 

Con el nodo “Value conter” podemos verificar que se ha realizado correctamente:



### RESULTADOS

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| KNN (NOR2) | 6554 | 2722 | 34433 | 5133 | 0,707 | 0,561 | 0,927 | 0,625 | 0,839 | 0,721 | 0,862 |
| KNN (ESS) | 9624 | 2695 | 8992 | 2063 | 0,781 | 0,823 | 0,769 | 0,802 | 0,796 | 0,796 | 0,864 |
| C4.5 (NOR2) | 7106 | 3088 | 34067 | 4581 | 0,697 | 0,608 | 0,917 | 0,65 | 0,843 | 0,747 | 0,877 |
| C4.5 (ESS) | 9718 | 2630 | 9057 | 1969 | 0,787 | 0,832 | 0,775 | 0,809 | 0,803 | 0,8083 | 0,871 |
| Logistic Regression (NOR2) | 6678 | 2630 | 34525 | 5009 | 0,717 | 0,571 | 0,929 | 0,636 | 0,844 | 0,729 | 0,897 |
| Logistic Regression (ESS) | 9940 | 2586 | 9101 | 1747 | 0,794 | 0,851 | 0,779 | 0,821 | 0,812 | 0,814 | 0,897 |
| Random Forest (NOR2) | 6266 | 1497 | 35658 | 5421 | 0,807 | 0,536 | 0,96 | 0,644 | 0,858 | 0,717 | 0,869 |
| Random Forest (ESS) | 10651 | 3379 | 8308 | 1036 | 0,759 | 0,911 | 0,711 | 0,828 | 0,811 | 0,805 | 0,892 |
| Gradient Boost (NOR2) | 7045 | 1970 | 35185 | 4642 | 0,781 | 0,603 | 0,94 | 0,681 | 0,865 | 0,756 | 0,921 |
| Gradient Boost (ESS) | 10250 | 2360 | 9327 | 1437 | 0,813 | 0,877 | 0,798 | 0,844 | 0,838 | 0,837 | 0,921 |
| Neural Networks (NOR2) | 6757 | 2290 | 34865 | 4930 | 0,747 | 0,578 | 0,938 | 0,652 | 0,852 | 0,737 | 0,908 |
| Neural Networks (ESS) | 10147 | 2655 | 9032 | 1540 | 0,793 | 0,868 | 0,773 | 0,829 | 0,821 | 0,819 | 0,905 |

#### ANÁLISIS

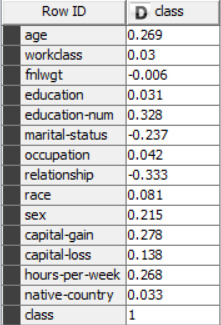
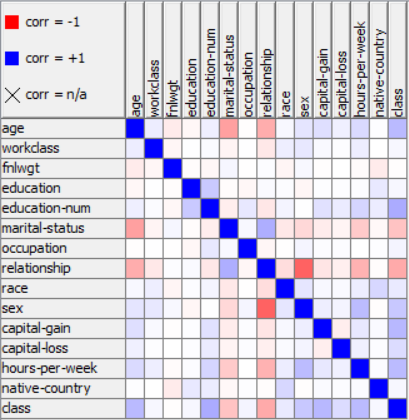
Se puede observar cómo en todos los algoritmos hemos perdido algo de precisión (accuracy), algo de acierto sobre la clase negativa(TNR), pero aumentado en acierto sobre la clase positiva(TPR), por tanto, mejoramos tanto G-mean como F1\_score.

Con esto hemos conseguido lo que esperábamos, equilibrar el aprendizaje.

Aunque con esto hemos podido observar la influencia de que las clases estén equilibradas, podemos hacer propuestas más interesantes que simplemente eliminar ejemplos, como crear nuevos ejemplos mediante submuestreo.

# 6. INTERPRETACIÓN DE RESULTADOS

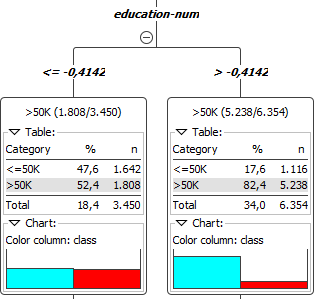




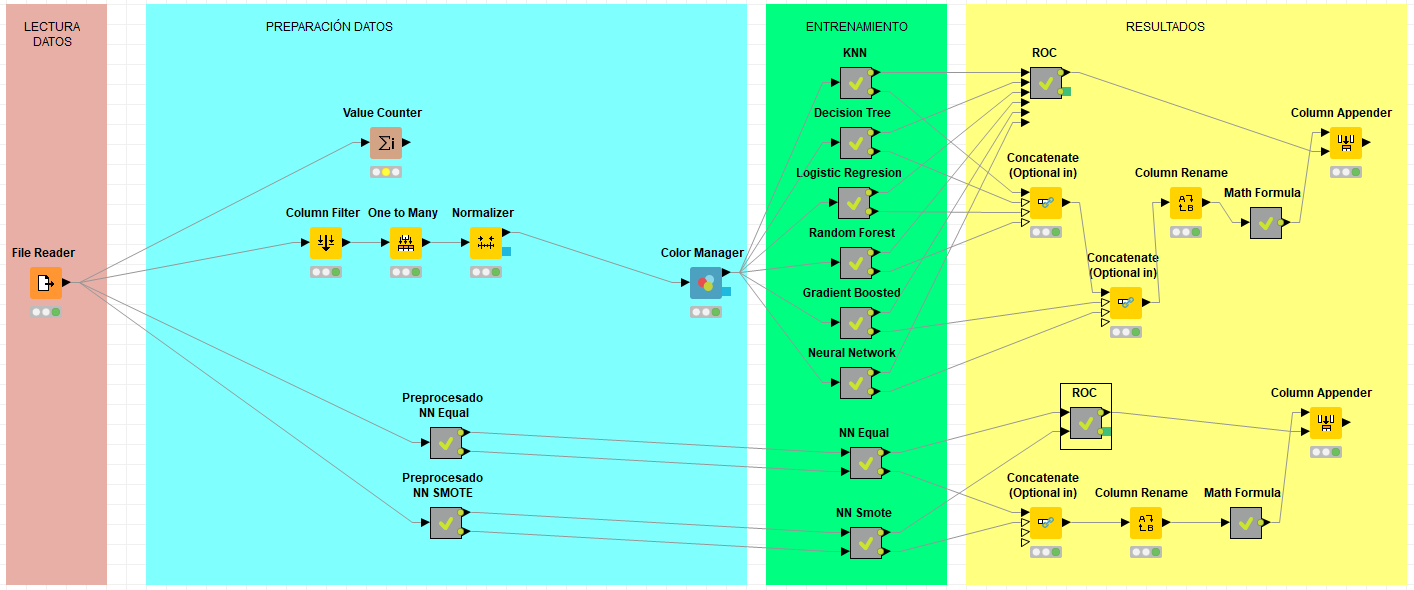
Según el árbol de decisión la variable más determinante para saber si ganará o no >50k es “relationship” que, si miramos en los valores de la correlación, vemos que también es la que mayor valor tiene en valor absoluto, por tanto, es la más influyente. Dentro de esta variable, el valor que más cantidad de ejemplos agrupa en el árbol de decisión es “=Husband”.

Si observamos la siguiente variable más relevante en la correlación, es “education-num”, que, además, dentro de la rama “=Husband” es la siguiente variable que más elementos separa.

Parece algo que cualquiera podría pensar de antemano, que la educación es importante en que pueda ganar más de 50.000$.



# 7. CONTENIDO ADICIONAL

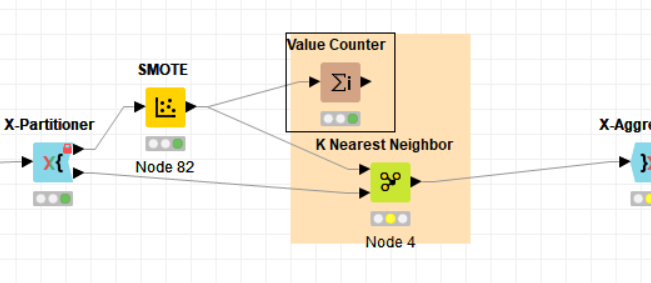


Se ha añadido las dos filas de abajo del flujo.

## 7.1. SUBMUESTREO

En esta ocasión vamos a utilizar el nodo “SMOTE” en vez del nodo “Equal size sampling”, pero no podemos submuestrear sobre datos de test, y hasta que hacemos la validación cruzada, no podemos saber que datos van a ser de train y cuáles de test.

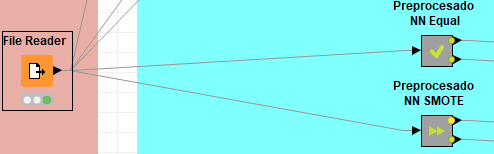
Incorporar “SMOTE” justo después de la salida de los datos de train en X-Partitioner (como hice con “Equal size sampling”) haría que submuestreara los datos 5 veces y eso tardaría mucho.

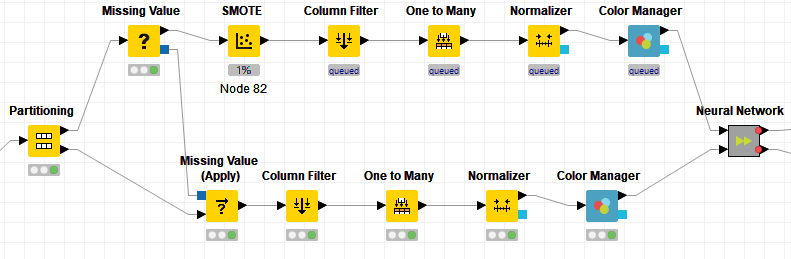


Por tanto, he optado por hacer la experimentación una sola vez sin validación cruzada y con el algoritmo “Neural Networks”.

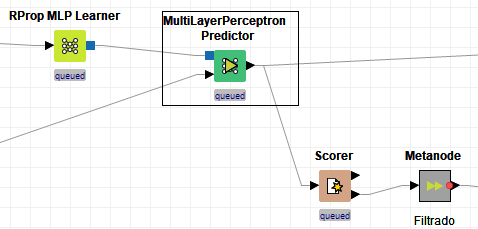
Aplicamos el mismo proceso en ambos, solamente cambiando el nodo “SMOTE” y “Equal size sampling”.

La opción elegida en SMOTE es “Oversample minority clases” que dejara las clases equilibradas con el mismo número de ejemplos.



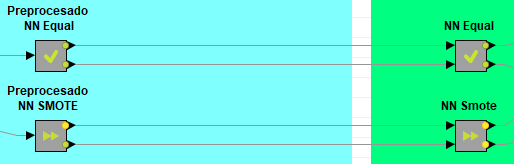




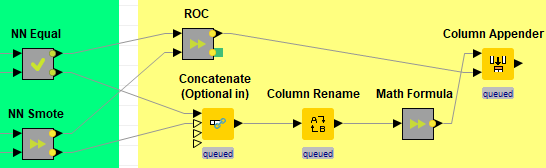


El flujo de nodos recibe los datos en “Partitioning”, que particiona los datos en 80%-20% para train y test.

Tanto a train como a test, le aplicamos el procesado de datos que hemos realizado durante la práctica, con la diferencia que para los datos de train, aplicamos el nodo “SMOTE” o “Equal size sampling”.



Por último, le pasamos los datos al algoritmo para el entrenamiento.



Una vez entrenado, comparamos los resultados.

### RESULTADOS

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | Iter/Layers | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| NN Equal | I100-L3 | 2173 | 2468 | 4982 | 146 | 0,468 | 0,937 | 0,669 | 0,624 | 0,732 | 0,792 | 0,898 |
| NN Smote | I100-L3 | 2207 | 2760 | 4690 | 112 | 0,444 | 0,952 | 0,630 | 0,606 | 0,706 | 0,774 | 0,886 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | Iter/Layers | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| NN Equal | I200-L3 | 2169 | 2475 | 4975 | 150 | 0,467 | 0,935 | 0,668 | 0,623 | 0,731 | 0,790 | 0,896 |
| NN Smote | I200-L3 | 2198 | 2828 | 4622 | 121 | 0,437 | 0,948 | 0,620 | 0,599 | 0,698 | 0,767 | 0,880 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | Iter/Layers | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| NN Equal | I100-L5 | 2175 | 2377 | 5073 | 144 | 0,478 | 0,938 | 0,681 | 0,633 | 0,742 | 0,799 | 0,897 |
| NN Smote | I100-L5 | 2215 | 2915 | 4535 | 104 | 0,432 | 0,955 | 0,609 | 0,595 | 0,691 | 0,763 | 0,880 |

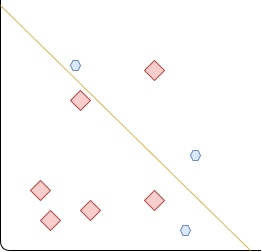
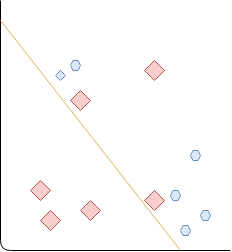
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row ID | Iter/Layers | TP | FP | TN | FN | PPV | TPR | TNR | F1\_score | Accuracy | G\_mean | Área ROC |
| NN Equal | I200-L5 | 2146 | 2203 | 5247 | 173 | 0,493 | 0,925 | 0,704 | 0,644 | 0,757 | 0,807 | 0,894 |
| NN Smote | I200-L5 | 2226 | 3373 | 4077 | 93 | 0,398 | 0,960 | 0,547 | 0,562 | 0,645 | 0,725 | 0,849 |

#### ANÁLISIS

De este experimento podemos sacar varias conclusiones.

Por un lado, vemos como con SMOTE se ajusta más aún a la clase positiva, por lo que podríamos plantearnos si quizás no interesa equilibrar tanto las clases para el aprendizaje.

Con SMOTE, estamos creando ejemplos artificiales basados en los ejemplos ya existentes, con lo que equilibramos en número de ejemplos las clases. Esto hace que, en el ajuste del algoritmo, este se desplace más hacia la clase positiva que cuando había menos ejemplos. Cuando recibe los datos de test, al estar más desplazado hacia la clase positiva, muchos de los ejemplos que antes caían del lado de los negativos, ahora caen en el lado de los positivos y se clasifican mal.

En estas dos imágenes vemos un ejemplo sencillo de esto. El algoritmo intenta acertar el máximo número de ejemplos, por tanto, en la primera imagen separa de tal forma que sólo falla en 2 de los ejemplos, la mejor opción posible con una línea recta. En la segunda imagen, se han creado ejemplos nuevos artificiales basados en los reales de la clase azul, y vemos como al intentar acertar el máximo número de ejemplos, desplaza la línea y provoca que dos de los ejemplos rojos que antes estaban bien clasificados, ahora no lo estén.

Por esto vemos en las tablas, que cuando sube el porcentaje en TPR, baja en TNR, que coincide con el aumento de las capas y el aumento de las iteraciones, porque el algoritmo está, ajustando cada vez más.

# 8.BIBLIOGRAFÍA

[https://docs.microsoft.com/es-es/azure////machine-learning/studio/algorithm-choice?toc=%2Fes-es%2Fazure%2F%2F%2F%2Fmachine-learning%2Fteam-data-science-process%2Ftoc.json&bc=%2Fes-es%2Fazure%2Fbread%2Ftoc.json](https://docs.microsoft.com/es-es/azure/machine-learning/studio/algorithm-choice?toc=%2Fes-es%2Fazure%2F%2F%2F%2Fmachine-learning%2Fteam-data-science-process%2Ftoc.json&bc=%2Fes-es%2Fazure%2Fbread%2Ftoc.json)

<http://sci2s.ugr.es/graduateCourses/in>