
Práctica-1: Análisis Predictivo Mediante Clasificación.

UNIVERSIDAD DE GRANADA
E.T.S.I. INFORMÁTICA Y TELECOMUNICACIÓN



**UNIVERSIDAD
DE GRANADA**

**Departamento de Ciencias de la
Computación e Inteligencia Artificial**

Inteligencia de Negocio (2019-2020)

Daniel Bolaños Martínez
danibolanos@correo.ugr.es
Grupo 2 - Jueves 09:30h

Índice

1. Introducción.	3
2. Resultados obtenidos.	4
2.1. Árbol de decisión: C4.5.	7
2.2. Gradient Boosted.	8
2.3. Naive Bayes.	9
2.4. Random Forest.	10
2.5. K-NN.	11
2.6. Red Neuronal: MLP.	12
3. Analisis de resultados.	13
4. Configuracion de algoritmos.	17
4.1. Árbol de decisión: C4.5.	17
4.2. Random Forest.	22
4.3. K-NN.	27
5. Procesado de datos.	34
5.1. Random Forest.	38
5.2. K-NN.	40
6. Interpretacion de resultados.	43
7. Contenido adicional.	45
8. Bibliografía.	45

1. Introducción.

El objetivo de la práctica es crear un sistema inteligente que pueda predecir con los datos de Taarifa y el Ministerio de Agua de Tanzania, qué bombas de extracción de agua funcionan, cuales necesitan algunas reparaciones y cuales no funcionan.

Para predecir una de estas tres clases, se dispone de 39 variables (numéricas y categóricas) que ofrecen información sobre que tipo de bomba se está usando, cuando se instaló, cómo se administra, su ubicación, datos sobre la cuenca geográfica, tipo de extracción, coste del agua, etc. Conociendo la predicción de las bombas que fallarán, podremos mejorar las operaciones de mantenimiento y garantizar la disponibilidad de agua potable en las comunidades de Tanzania. El conjunto de datos consta de 59.400 instancias las cuales algunas cuentan con valores desconocidos.

Los datos están basados en la competición *Pump it Up: Data Mining the Water Table* disponible en: <https://www.drivedata.org/competitions/7/pump-it-up-data-mining-the-water-table/>.

A lo largo de la práctica, estudiaremos el comportamiento de diferentes algoritmos de clasificación sobre los datos propuestos, haremos una comparación de los mismos y extraeremos conclusiones. Utilizaremos 6 algoritmos distintos, sobre los que realizaremos una validación cruzada de 5 particiones con muestreo estratificado, tomando como semilla aleatoria mi DNI.

Los 6 algoritmos elegidos han sido:

- C4.5 (Decision Tree)
- Gradient Boosted
- Naive Bayes
- Random Forest
- K-NN (K Nearest Neighbor)
- MLP (Neural Network)

Realizaré un análisis exhaustivo con distintos parámetros sobre los algoritmos: C4.5, Random Forest y K-NN para estudiar como se comportan frente a distintas modificaciones de sus valores. Se extraerán tablas comparativas para los distintos algoritmos así como sus curvas ROC, diagramas de barras y cualquier tipo de gráfica que pueda mejorar la observación de los resultados obtenidos, así como su comparación posterior.

Finalmente, haré un preprocesado del dataset y probaré su funcionamiento sobre los algoritmos anteriores para comparar sus resultados.

2. Resultados obtenidos.

Ejecutaremos cada algoritmo de forma individual y mostraremos su tabla de resultados. En este apartado utilizaremos el preprocesado básico que nos permita ejecutar los algoritmos sin errores. Presentaré, para cada algoritmo: el flujo de trabajo de *KNIME*, los resultados básicos de la tabla para cada algoritmo sobre los datos de test y su matriz de confusión. El estudio de la curva ROC y el overfitting se dejará para el apartado 3: Análisis de los Resultados.

Empezaremos leyendo el archivo *water_pump.csv* que contiene los datos del problema, añadimos un color a las filas para distinguir la clase a la que pertenece cada instancia. Y finalmente realizamos en cada nodo la validación cruzada que nos permitirá obtener los resultados de cada algoritmo.

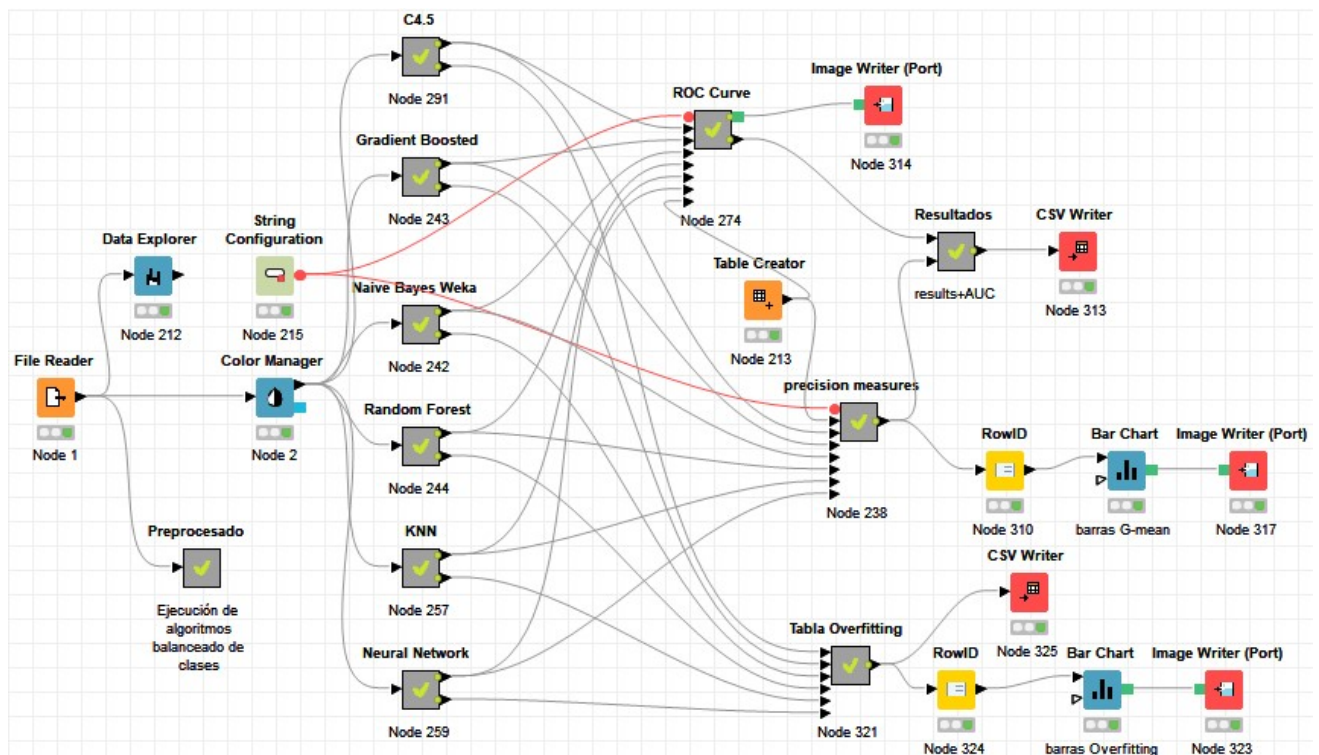
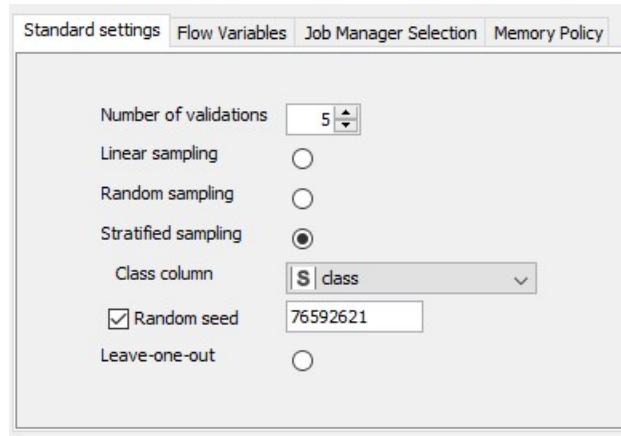


Figura 1: Flujo de trabajo de KNIME.

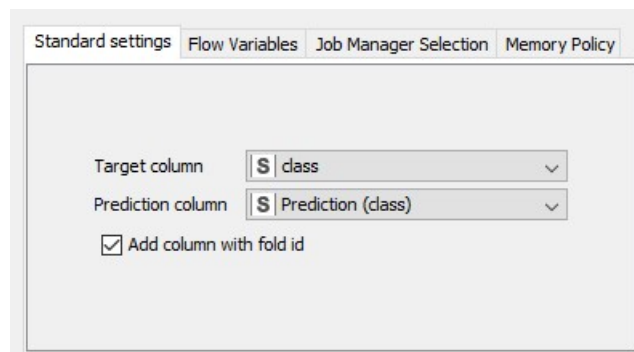
Se mostrará el flujo de trabajo que contiene la validación cruzada para cada algoritmo. De forma común, los algoritmos tendrán la siguiente configuración para el nodo de particionado de los datos y para el de agregado de los mismos:



The screenshot shows the 'X-Partitioner' configuration window with the 'Flow Variables' tab selected. The settings are as follows:

- Number of validations: 5 (spin box)
- Linear sampling: ☐
- Random sampling: ☐
- Stratified sampling: ☒
- Class column: S | class (dropdown menu)
- ☒ Random seed: 76592621 (text box)
- Leave-one-out: ☐

Figura 2: Configuración X-Partitioner.



The screenshot shows the 'X-Aggregator' configuration window with the 'Flow Variables' tab selected. The settings are as follows:

- Target column: S | class (dropdown menu)
- Prediction column: S | Prediction (class) (dropdown menu)
- ☒ Add column with fold id

Figura 3: Configuración X-Aggregator.

Con esta configuración nos aseguramos de que se hagan 5 validaciones y un muestreo estratificado sobre las clases de la misma. Utilizando mi número de DNI como semilla del muestreo aleatorio. Finalmente obtendremos la predicción de la clase en la columna con su mismo nombre.

Este primer apartado, se hace con el mínimo preprocesado que necesitan los algoritmos para ejecutar correctamente el conjunto de datos. Algunos de ellos daban un *warning* que advertía acerca de la omisión de algunas columnas o valores perdidos,

por lo que para evitarlo, he optado por hacer un **Column filter** sobre las columnas omitidas o un **Missing Value** sobre los algoritmos que lo necesiten.

En el caso de que diesen problemas con las columnas, siempre eran las mismas, ya que en un estudio sobre el conjunto de categorías sobre esas columnas utilizando **Data Explorer**, se puede observar que superan las 999 valores. Para el caso de los valores perdidos se ha optado por sustituir los numéricos por la media y los categóricos por el valor más frecuente.

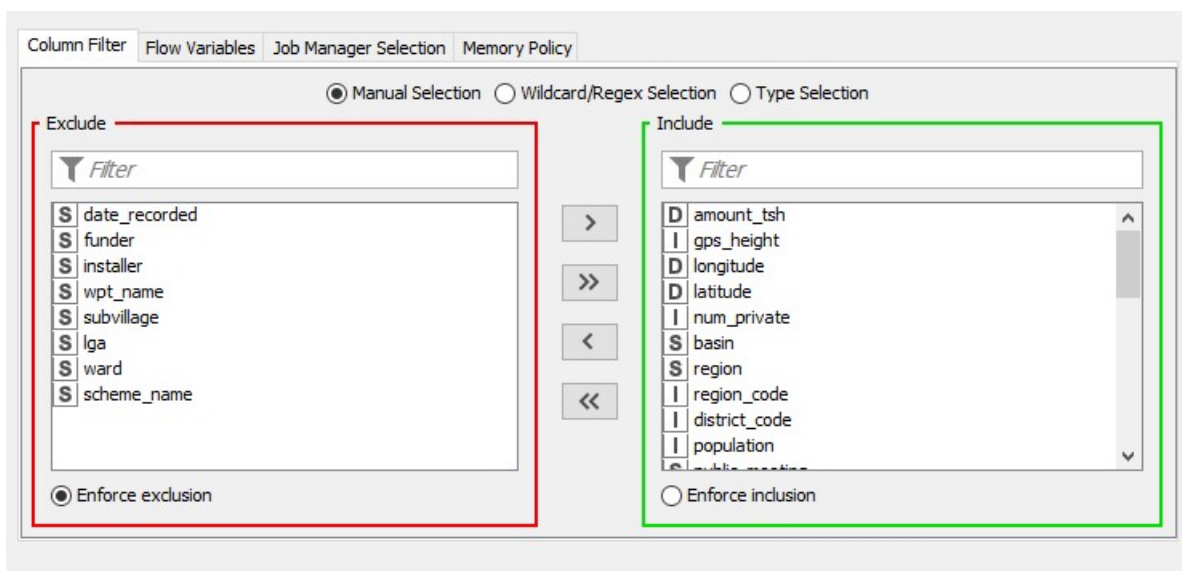


Figura 4: Configuración Column Filter.

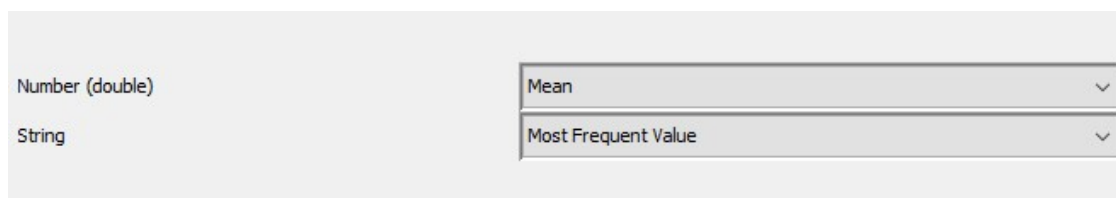


Figura 5: Configuración Missing Values.

2.1. Árbol de decisión: C4.5.

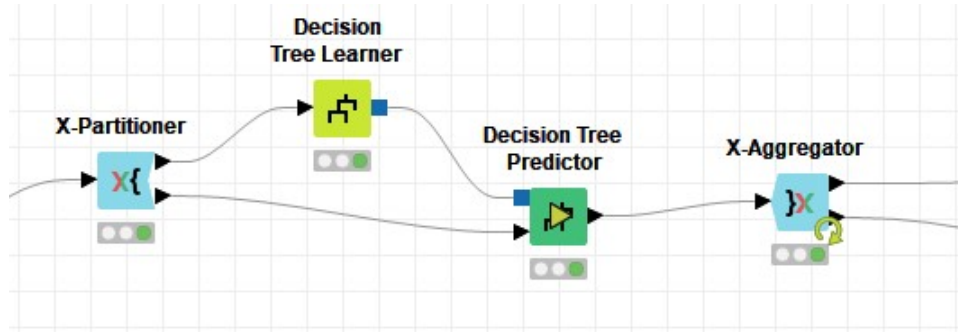


Figura 6: Cross-Validation C4.5 (Decision Tree).

El nodo **Decision Tree Learner** a partir de los datos de entrenamiento, induce un árbol de decisión que será usado para clasificar los datos. Por otra parte, el nodo **Decision Tree Predictor** obtiene el árbol de decisión del Learner así como el conjunto de datos de test, de esta forma podrá predecir el valor de la clase de las muestras de prueba. El nodo **X-Aggregator** se encarga de obtener la tabla de predicciones.

Para todos los algoritmos, a partir de la tabla de predicciones, generaremos usando el nodo **Scorer**, la matriz de confusión y las estadísticas del algoritmo. A partir de ellos calcularemos con el nodo **Math Formula**, los valores de $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$ y $G-mean = \sqrt{TPR \cdot TNR}$ que serán útiles a la hora de comparar los algoritmos.

Name	Accuracy	AUC	F1-score	FN	FP	G-mean	PPV	TN	TNR	TP	TPR
C4.5	0.826	0.841	0.771	5373	4971	0.813	0.778	31605	0.864	17451	0.765

Tabla 1: Datos test obtenidos para C4.5 (Decision Tree).

	functional	non functional	functional needs repair	error
functional	26469	4208	1336	246
non functional	4601	17451	544	228
functional needs repair	2015	763	1495	44

Tabla 2: Matriz de confusión de C4.5 (Decision Tree).

2.2. Gradient Boosted.

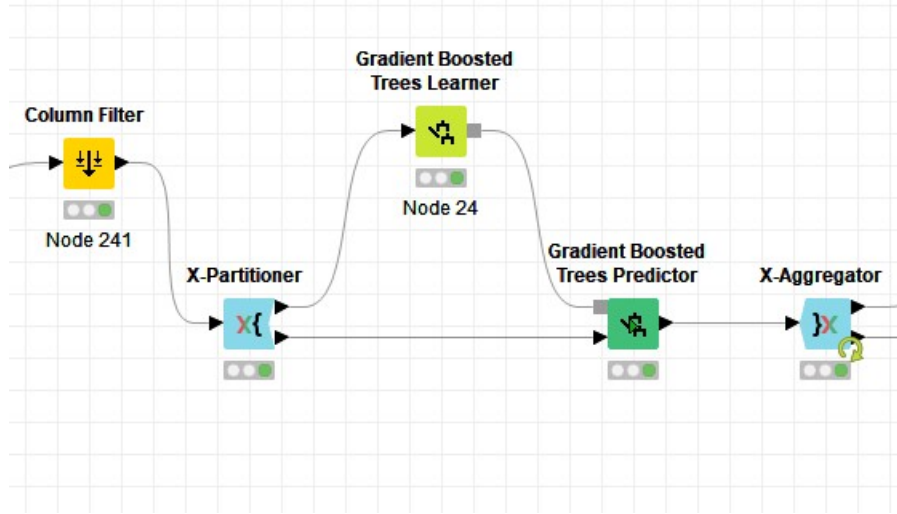


Figura 7: Cross-Validation Gradient Boosted.

El nodo **Gradient Boosted Learner** a partir de los datos de entrenamiento, utiliza árboles de regresión para clasificar los datos. Por otra parte, el nodo **Gradient Boosted Trees Predictor** obtiene el modelo del Learner así como el conjunto de datos de test, de esta forma podrá predecir el valor de la clase de las muestras de prueba. El nodo **X-Aggregator** se encarga de obtener la tabla de predicciones.

A partir de la tabla de predicciones, generaremos a partir del nodo **Scorer**, la matriz de confusión y las estadísticas del algoritmo.

Name	Accuracy	AUC	F1-score	FN	FP	G-mean	PPV	TN	TNR	TP	TPR
Gradient Boosted	0.825	0.889	0.75	7191	3206	0.791	0.83	33370	0.912	15633	0.685

Tabla 3: Datos test obtenidos para Gradient Boosted.

	functional	non functional	functional needs repair
functional	29502	2502	255
non functional	7025	15633	166
functional needs repair	2839	704	774

Tabla 4: Matriz de confusión de Gradient Boosted.

2.3. Naive Bayes.

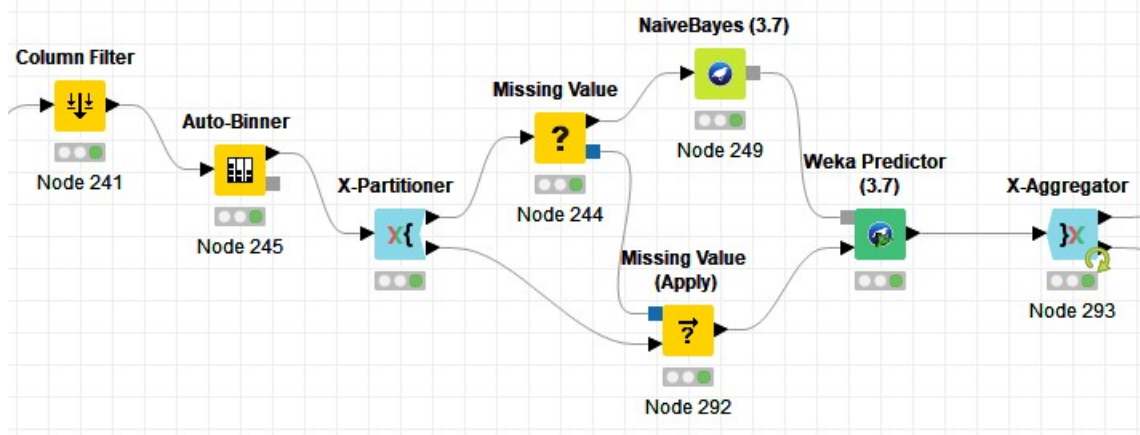


Figura 8: Cross-Validation Naive Bayes.

Este algoritmo está basado en un clasificador probabilístico que se apoya en los resultados del Teorema de Bayes. El nodo **NaiveBayes (3.7)** a partir de los datos de entrenamiento, genera un modelo bayesiano. Por otra parte, el nodo **Weka Predictor (3.7)** obtiene el modelo así como el conjunto de datos de test, de esta forma podrá predecir el valor de la clase de las muestras de prueba. El nodo **X-Aggregator** se encarga de obtener la tabla de predicciones.

A partir de la tabla de predicciones, generaremos a partir del nodo **Scorer**, la matriz de confusión y las estadísticas del algoritmo.

Name	Accuracy	AUC	F1-score	FN	FP	G-mean	PPV	TN	TNR	TP	TPR
Naive Bayes	0.759	0.8	0.655	9248	5085	0.716	0.728	31491	0.861	13576	0.595

Tabla 5: Datos test obtenidos para Naive Bayes.

	functional	non functional	functional needs repair
functional	24364	4340	3555
non functional	7452	13576	1796
functional needs repair	2017	745	1555

Tabla 6: Matriz de confusión de Naive Bayes.

2.4. Random Forest.

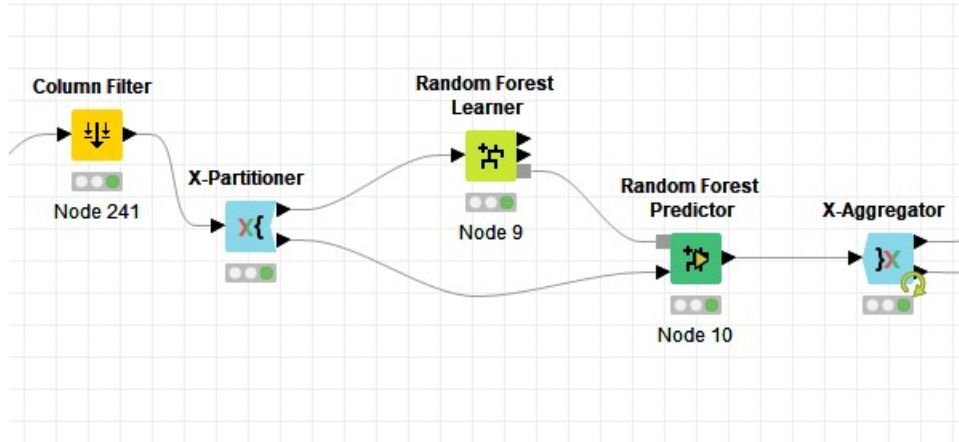


Figura 9: Cross-Validation Random Forest.

El nodo **Random Forest Learner** a partir de los datos de entrenamiento, genera un modelo de bosque aleatorio de árboles de decisión. Por otra parte, el nodo **Random Forest Predictor** obtiene el modelo del Learner así como el conjunto de datos de test, de esta forma podrá predecir el valor de la clase de las muestras de prueba. El nodo **X-Aggregator** se encarga de obtener la tabla de predicciones.

A partir de la tabla de predicciones, generaremos a partir del nodo **Scorer**, la matriz de confusión y las estadísticas del algoritmo.

Name	Accuracy	AUC	F1-score	FN	FP	G-mean	PPV	TN	TNR	TP	TPR
Random Forest	0.858	0.92	0.803	5634	2796	0.834	0.86	33780	0.924	17190	0.753

Tabla 7: Datos test obtenidos para Random Forest.

	functional	non functional	functional needs repair
functional	29517	2211	531
non functional	5353	17190	281
functional needs repair	2509	585	1223

Tabla 8: Matriz de confusión de Random Forest.

2.5. K-NN.

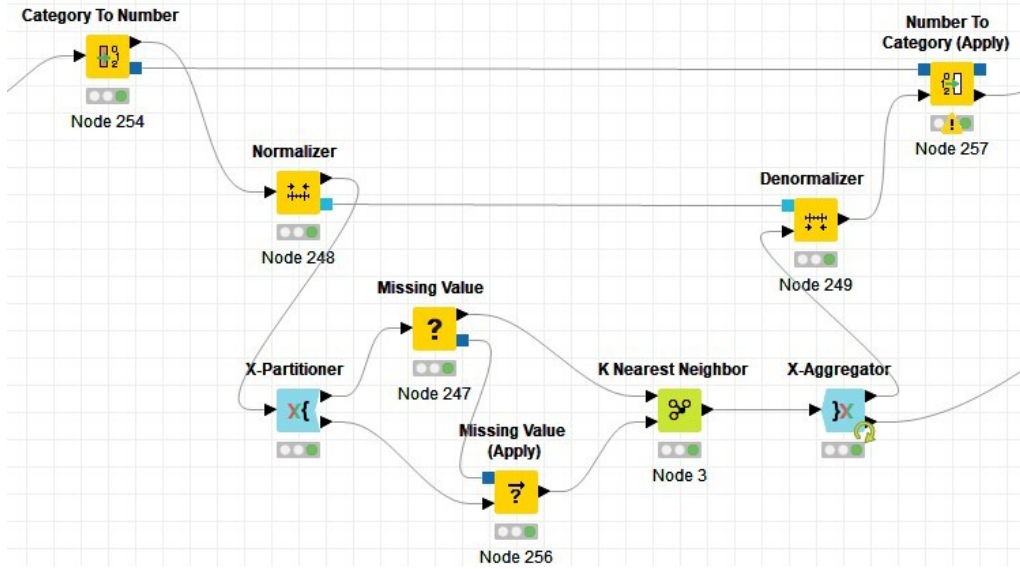


Figura 10: Cross-Validation K-NN.

El algoritmo K-NN se basa en el cálculo de la distancia de cada elemento del conjunto de datos sobre sus vecinos y selecciona los K vecinos más cercanos. En este caso, el nodo **K Nearest Neighbor** tiene dos puertos: uno para train y otro para test. Por lo que internamente realizará el cálculo del modelo y la predicción de los datos de forma conjunta. El algoritmo opera con valores numéricos por lo que es necesario pasar los categóricos a número y normalizarlos para obtener mejores resultados.

A partir de la tabla de predicciones que proporciona **X-Aggregator**, generaremos a partir del nodo **Scorer**, la matriz de confusión y las estadísticas del algoritmo.

Name	Accuracy	AUC	F1-score	FN	FP	G-mean	PPV	TN	TNR	TP	TPR
KNN	0.798	0.841	0.727	6884	5101	0.775	0.758	31475	0.861	15940	0.698

Tabla 9: Datos test obtenidos para K-NN.

	functional	non functional	functional needs repair
functional	26939	4325	995
non functional	6382	15940	502
functional needs repair	2344	776	1197

Tabla 10: Matriz de confusión de K-NN.

2.6. Red Neuronal: MLP.

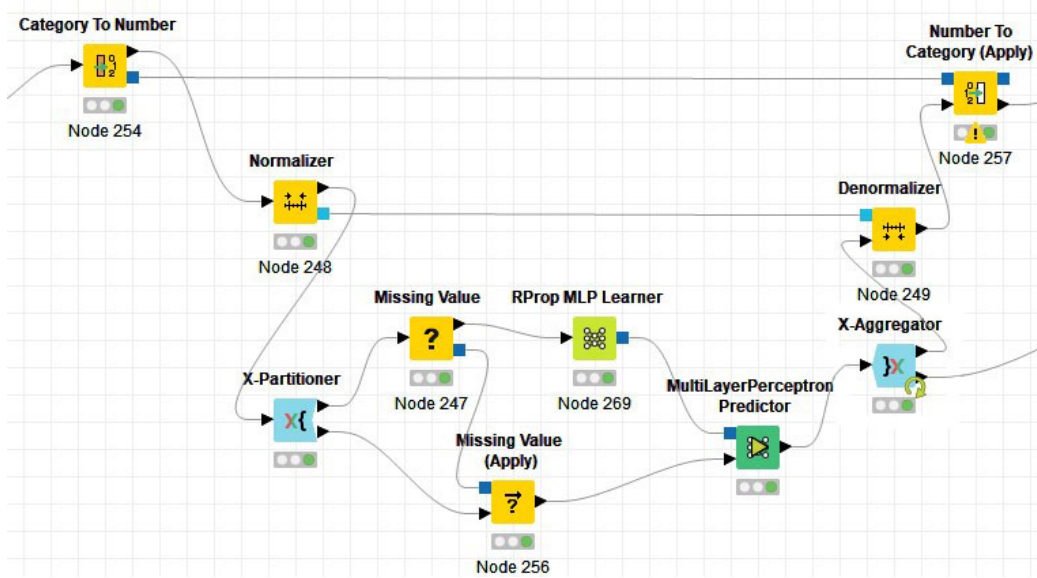


Figura 11: Cross-Validation MLP (Neural Network).

El algoritmo construye un modelo de unidades neuronales conectadas entre sí por enlaces. El nodo **Prop MLP Learner** contruye el modelo a partir de los datos de test y **MultiLayerPerceptron Predictor** predice a partir de los de test y el modelo del Learner. El algoritmo opera con valores numéricos por lo que es necesario pasar los categóricos a número y normalizarlos para obtener mejores resultados.

A partir de la tabla de predicciones que proporciona **X-Aggregator**, generaremos a partir del nodo **Scorer**, la matriz de confusión y las estadísticas del algoritmo.

Name	Accuracy	AUC	F1-score	FN	FP	G-mean	PPV	TN	TNR	TP	TPR
MLP	0.751	0.799	0.652	8952	5859	0.714	0.703	30717	0.84	13872	0.608

Tabla 11: Datos test obtenidos para MLP (Neural Network).

	functional	non functional	functional needs repair
functional	27247	4952	60
non functional	8909	13872	43
functional needs repair	3287	907	123

Tabla 12: Matriz de confusión de MLP (Neural Network).

3. Análisis de resultados.

Haremos un análisis comparativo de los 6 algoritmos descritos anteriormente apoyándonos sobre la tabla general de los resultados, la curva ROC y los diferentes diagramas de barras obtenidos.

Name	Accuracy	AUC	F1-score	FN	FP	G-mean	PPV	TN	TNR	TP	TPR
C4.5	0.826	0.841	0.771	5373	4971	0.813	0.778	31605	0.864	17451	0.765
Gradient Boosted	0.825	0.889	0.75	7191	3206	0.791	0.83	33370	0.912	15633	0.685
Naive Bayes	0.759	0.8	0.655	9248	5085	0.716	0.728	31491	0.861	13576	0.595
Random Forest	0.858	0.92	0.803	5634	2796	0.834	0.86	33780	0.924	17190	0.753
KNN	0.798	0.841	0.727	6884	5101	0.775	0.758	31475	0.861	15940	0.698
MLP	0.751	0.799	0.652	8952	5859	0.714	0.703	30717	0.84	13872	0.608

Tabla 13: Tabla comparativa para los algoritmos descritos.

Primeramente, haremos una breve descripción de los valores usados en la tabla, que se corresponden con las medidas vistas en teoría.

- **Accuracy:** número de predicciones correctas entre el total de predicciones. Mide la tasa de aciertos respecto al total.
- **AUC:** área bajo la curva ROC. Permite representar en un único valor el rendimiento del clasificador.
- **FN y FP:** número de falsos negativos y falsos positivos respectivamente.
- **TN y TP:** número de verdaderos negativos y verdaderos positivos respectivamente.
- **PPV:** predicciones positivas correctas entre el número total de predicciones positivas.
- **TNR:** predicciones negativas correctas entre el número total de negativos.
- **TPR:** predicciones positivas correctas entre el número total de positivos.
- **F1-score:** media armónica de PPV y TPR. Penaliza más los errores al clasificar ejemplos positivos (FN) que los errores al clasificar ejemplos negativos (FP).
- **G-mean:** media geométrica de TPR y TNR. Trata de maximizar el acierto en ambas clases con un buen balance entre ambas. Es ambiguo debido que en ciertos problemas puede que consideremos más relevantes los errores en una clase que en otra.

A continuación analizaremos las medidas apoyándonos en gráficos de barras. Empezando por el valor de *Accuracy* de los algoritmos.

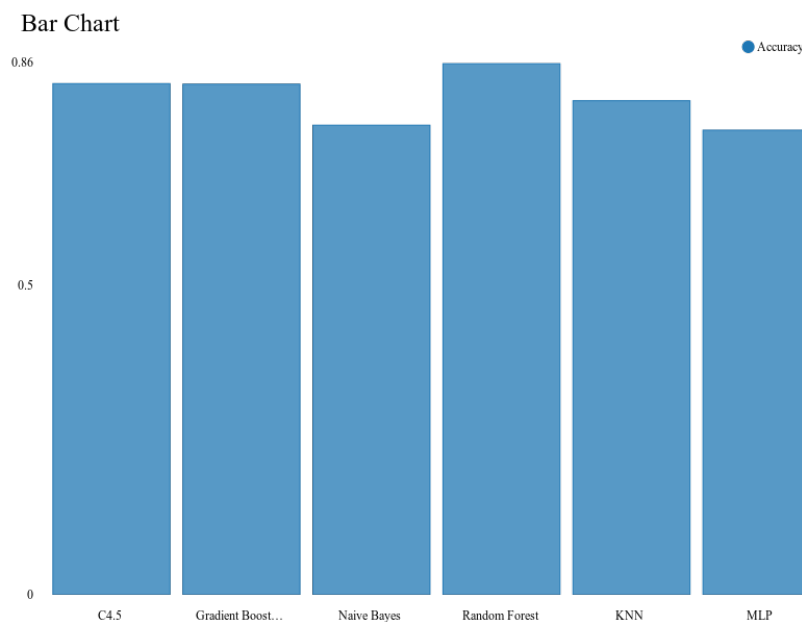


Figura 12: Gráfico de Barras Accuracy.

Los algoritmos que obtienen peores resultados son **Naive Bayes** que parte del supuesto de que las variables del problema son independientes, cosa que como estudiaremos más adelante no es así y **Neural Network** que quizás necesitaría un mayor número de casos de entrenamiento para mejorar su tasa de acierto, pero esto también elevaría el tiempo de ejecución.

Con unos resultados muy similares, destacan **C4.5** y **Gradient Boosted** dos algoritmos que trabajan con árboles. En general, ambos generan modelos sencillos, expresivos y que trabajan bien con valores categóricos, por lo que se adaptan bien al problema. En el caso del **K-NN**, obtenemos una tasa muy cercana a los dos anteriores. En esta versión, usamos $k = 3$ y realizamos un normalizado de los datos no obtenemos una mala tasa de clasificación, pero veremos que podemos configurarlo para mejorar su rendimiento.

Finalmente, el que mejor resultado ofrece, con un valor del 0.858 es **Random Forest**, que amplía el concepto del clasificador con árboles y ofrece mejores resultados que

C4.5 y Gradient Boosted.

En la Gráfica 13 analizaremos la diferencia entre los falsos positivos y los verdaderos positivos. Un algoritmo será mejor si es capaz de incrementar TP a un ritmo mucho mayor que FP.

El que mejor balance consigue es **Random Forest**, lo que nos está dando pistas de que será probablemente uno de los algoritmos sobre el que intentaremos mejorar sus resultados en el preprocesado.

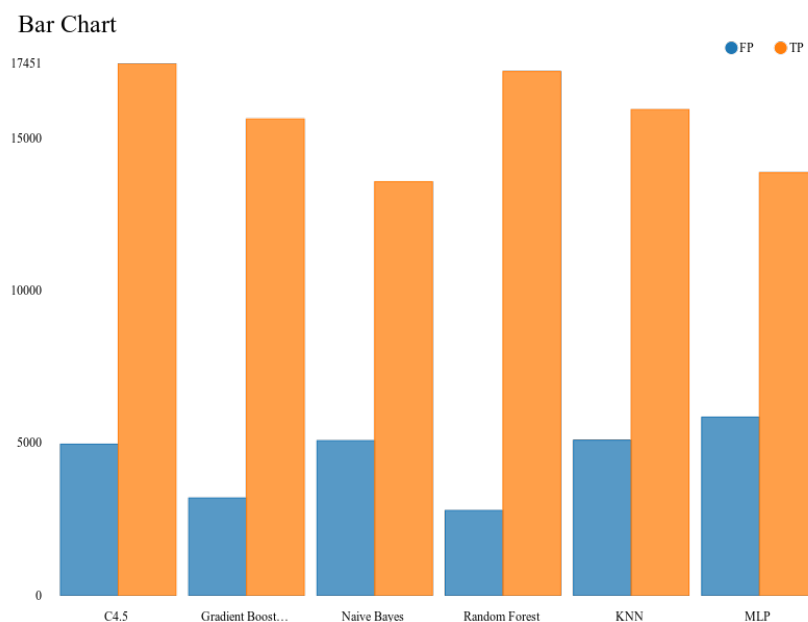


Figura 13: Gráfico de Barras FP-TN.

En la Gráfica 14 se comparan las medidas G-mean y F1-score. F1-score aumentará si tenemos un FN muy alto en comparación con FP. Con G-mean, se maximiza el balance de TPR y TNR. **Random Forest** vuelve a destacar de nuevo y el resto de algoritmos se mantienen en unos resultados similares a los que ya observamos en la gráfica de Accuracy.

Finalmente en la Gráfica 15 se muestra la curva ROC de los 6 algoritmos propuestos. El área bajo la curva nos indica el rendimiento del clasificador, que tiene una correlación directa con las medidas estudiadas anteriormente. Al igual que G-mean, maximiza el balance entre TNR y TPR.

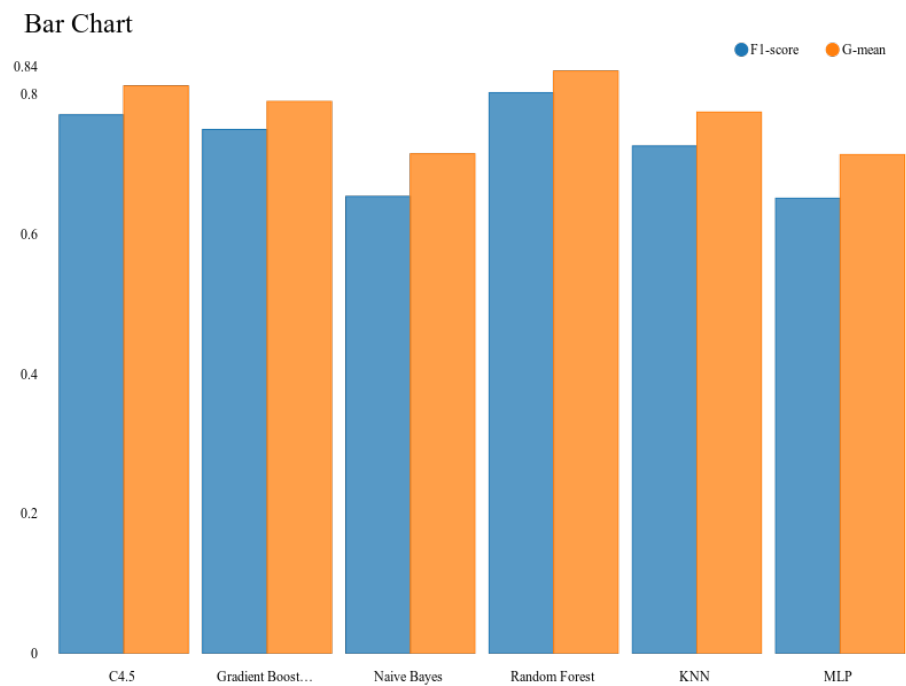


Figura 14: Gráfico de Barras G-mean/F1-score.

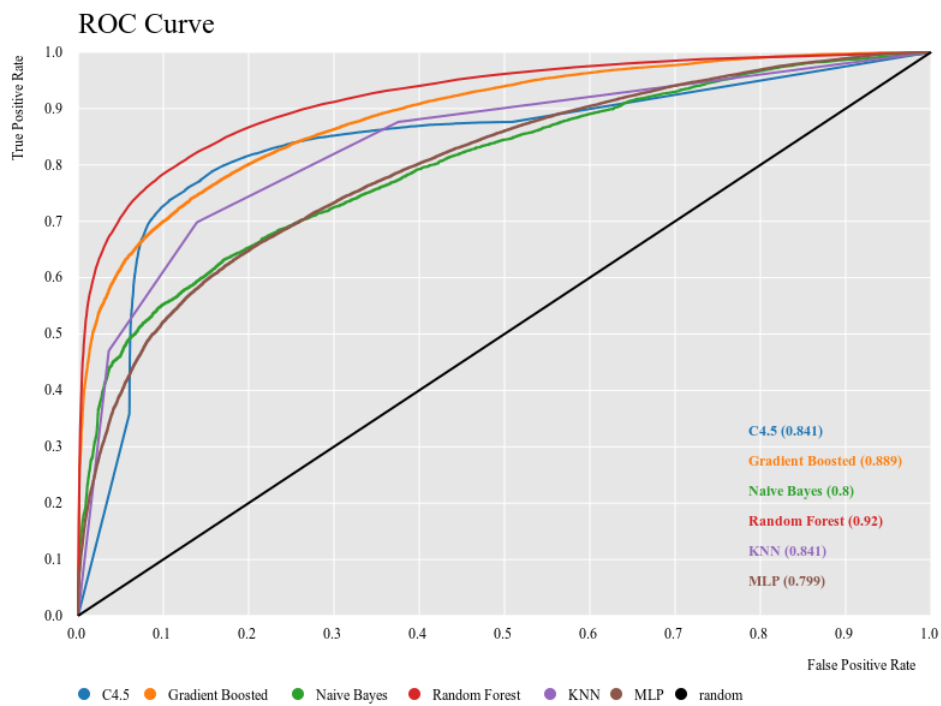


Figura 15: Curva ROC de los algoritmos.

4. Configuración de algoritmos.

En este apartado haremos un análisis más profundo sobre los algoritmos **C4.5**, **Random Forest** y **K-NN** incluyendo diferentes parámetros en la configuración de los algoritmos, un estudio del sobreaprendizaje de los mismos y en el caso de **C4.5**, la complejidad del modelo.

Empezaremos mostrando la configuración del entorno de trabajo de KNIME que se ha llevado a cabo para los 3 algoritmos elegidos. Obviaremos la configuración tanto de precision measures como de las tablas y ROC, ya que se ha realizado siguiendo los tutoriales de la Bibliografía.[2]

4.1. Árbol de decisión: C4.5.

En el caso del **C4.5**, hemos realizado 3 modificaciones además de la utilizada por defecto en la comparación inicial. Además obtenemos el tamaño del árbol de decisión en cada caso y un estudio del sobreaprendizaje del mismo.

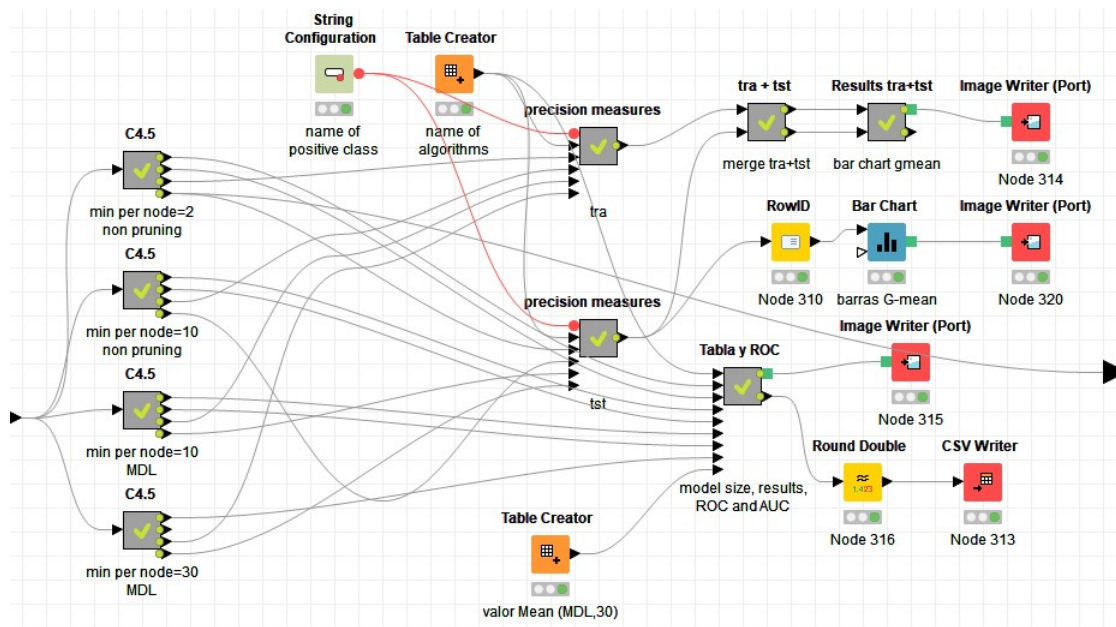


Figura 16: Workflow C4.5.

Mostraremos el contenido de uno de los nodos de **C4.5** y cómo hemos configurado cada uno de ellos, así como la obtención del tamaño de modelo. Para la configuración de train-test simplemente se ha añadido un nodo Predictor extra que predice las clases con los mismos datos de entrenamiento.

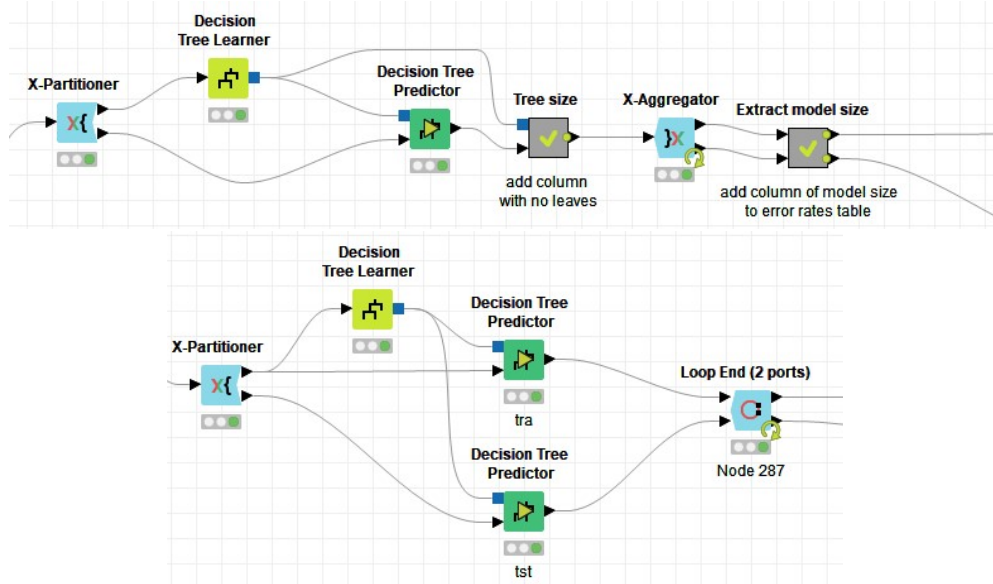


Figura 17: Interior Nodo C4.5.

Para la modificación de los parámetros, se ha accedido a la configuración del nodo **Decision Tree Learner** y se ha modificado la información relacionada con el método de poda que se puede elegir entre (no podar o MDL) y el número mínimo de registros por nodo, que por defecto viene como 2.

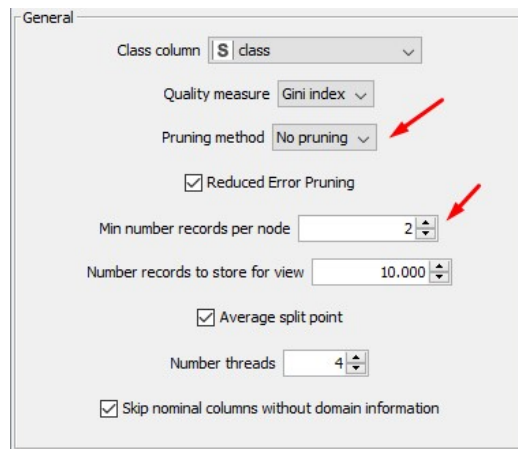


Figura 18: Configuración Decision Tree Learner.

Crearemos 4 metanodos y obtendremos información para:

- C4.5 con mínimo 2 registros por nodos y sin poda.
- C4.5 con mínimo 10 registros por nodos y sin poda.
- C4.5 con mínimo 10 registros por nodos y con poda MDL.
- C4.5 con mínimo 30 registros por nodos y con poda MDL.

Extraeremos una tabla para valorar los resultados y dibujaremos la curva ROC, para valorar como mejora o empeora el rendimiento del clasificador para las modificaciones realizadas.

Name	TP	FP	TN	FN	PPV	TPR	TNR	F1-score	Accuracy	G-mean	AUC	Model size	OF
(2,non)	17451	4971	31605	5373	0.778	0.765	0.864	0.771	0.826	0.813	0.841	6359.8	1.164
(10,non)	17095	4400	32176	5729	0.795	0.749	0.88	0.771	0.829	0.812	0.887	2275	1.064
(10,MDL)	16162	3779	32797	6662	0.81	0.708	0.897	0.756	0.824	0.797	0.871	1006.2	1.031
(30,MDL)	15680	3819	32757	7144	0.804	0.687	0.896	0.741	0.815	0.784	0.863	717.2	1.021

Tabla 14: Tabla comparativa modificaciones C4.5.

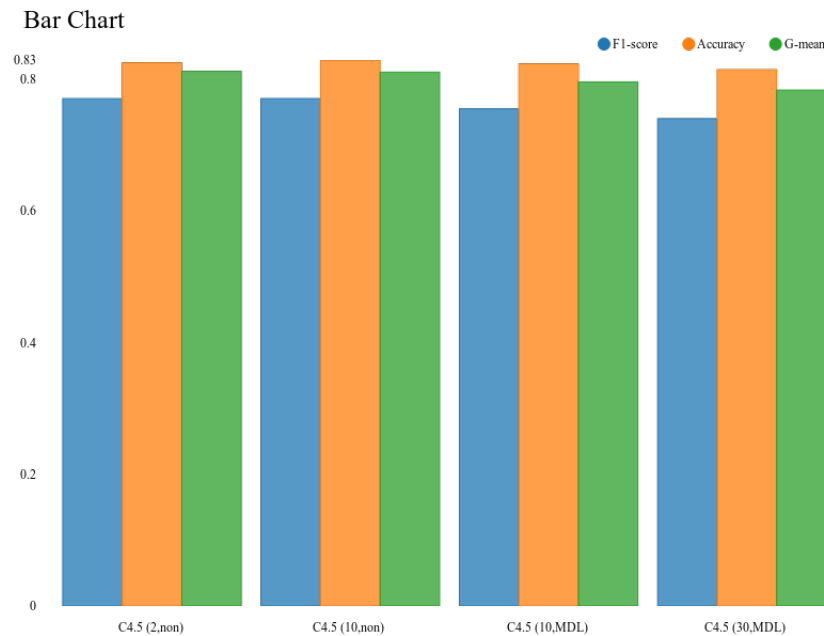


Figura 19: Gráfico de barras medidas C4.5.

En la Gráfica 19 podemos observar como los valores varían pocas milésimas entre una modificación y otra. En la Gráfica 20 se puede observar como al aumentar el mínimo de registros por nodo de 2 a 10 el tamaño del árbol de decisión disminuye más de la mitad, si realizamos la modificación de permitir poda, manteniendo los registros por nodo fijos, de nuevo reducimos el tamaño del árbol a la mitad.

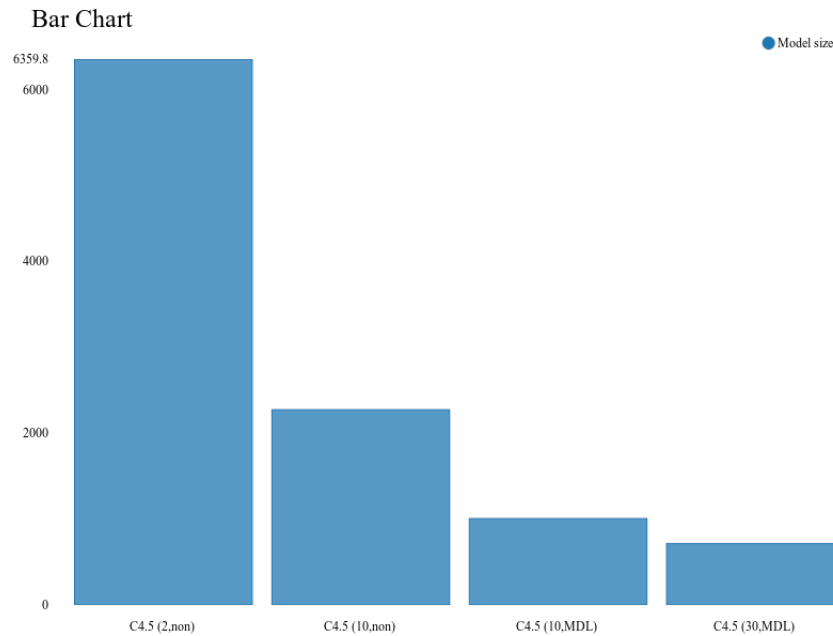


Figura 20: Gráfico de barras model size C4.5.

A continuación mostraremos el cálculo de G-mean sobre los conjuntos de train-test para valorar el sobreaprendizaje de cada modificación y ver cómo se comporta el algoritmo frente a esto.

En la Gráfica 21, podemos ver como la configuración por defecto estaba sobreaprendiendo, adaptándose demasiado a los datos de entrenamiento. Simplificando el modelo usando la poda, como hemos visto antes en la complejidad del modelo, obtenemos un overfitting más próximo a 1 y por tanto un clasificador más balanceado.

La Gráfica 22 muestra la curva ROC de los 4 casos y podemos observar qué modificación ofrece mejor rendimiento. La segunda modificación aumenta la precisión, teniendo un mejor balance de TNR y TPR, aunque viendo su overfitting, quizás de media el mejor algoritmo sea el que utiliza poda MDL con 10 registros por nodo, ya que sin usar poda tenemos un mayor sobreaprendizaje y aumentando el número de registros, empeoramos el rendimiento.

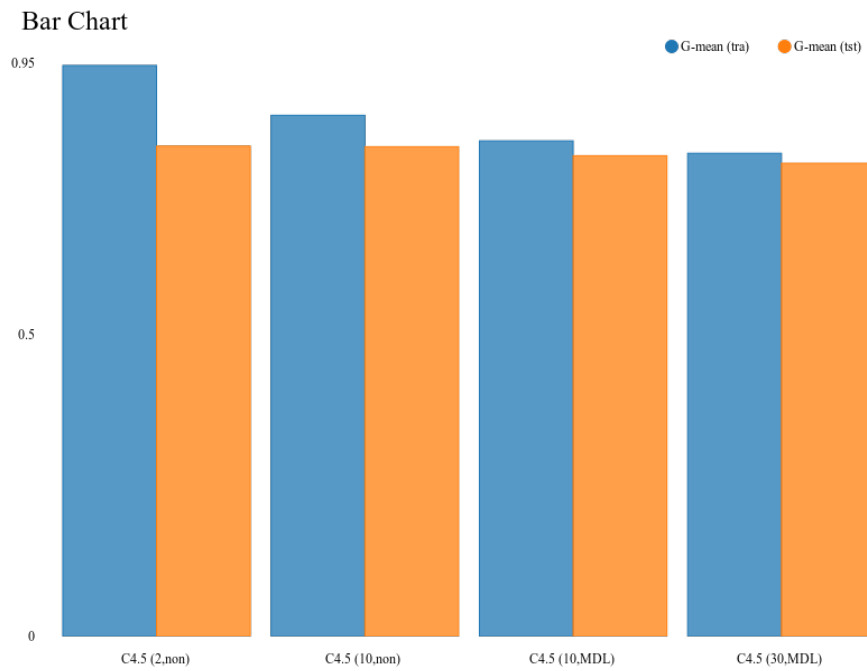


Figura 21: Gráfico de barras G-mean (train-test) C4.5.

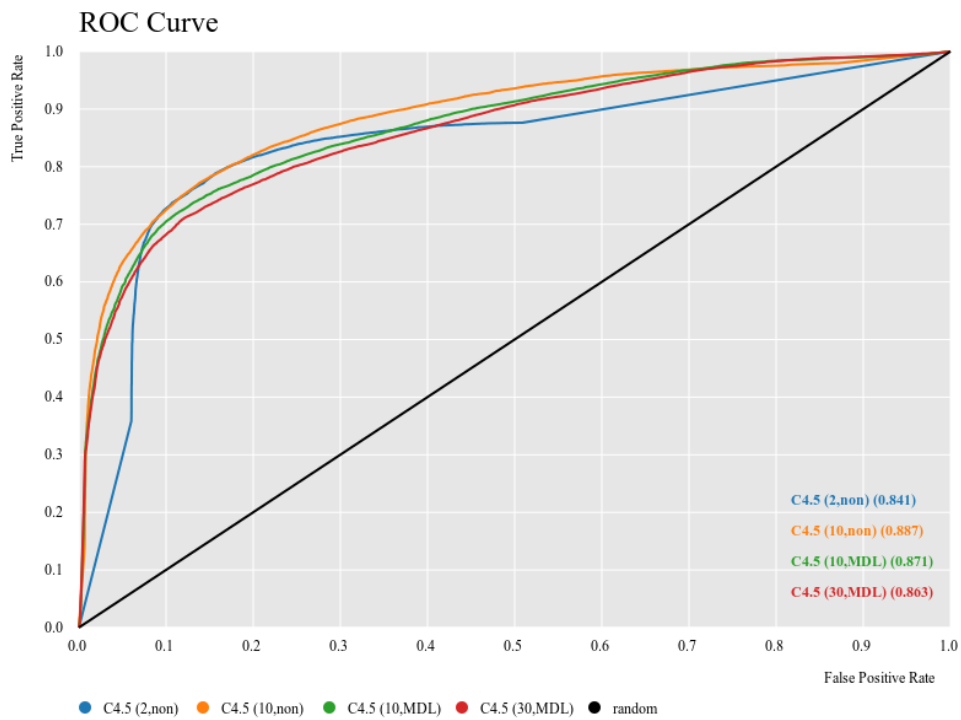


Figura 22: Curva ROC modificaciones C4.5.

4.2. Random Forest.

Para el **Random Forest** hemos realizado 4 modificaciones además de la utilizada por defecto en la comparación inicial. Además en cada caso realizamos un estudio del sobreaprendizaje del mismo.

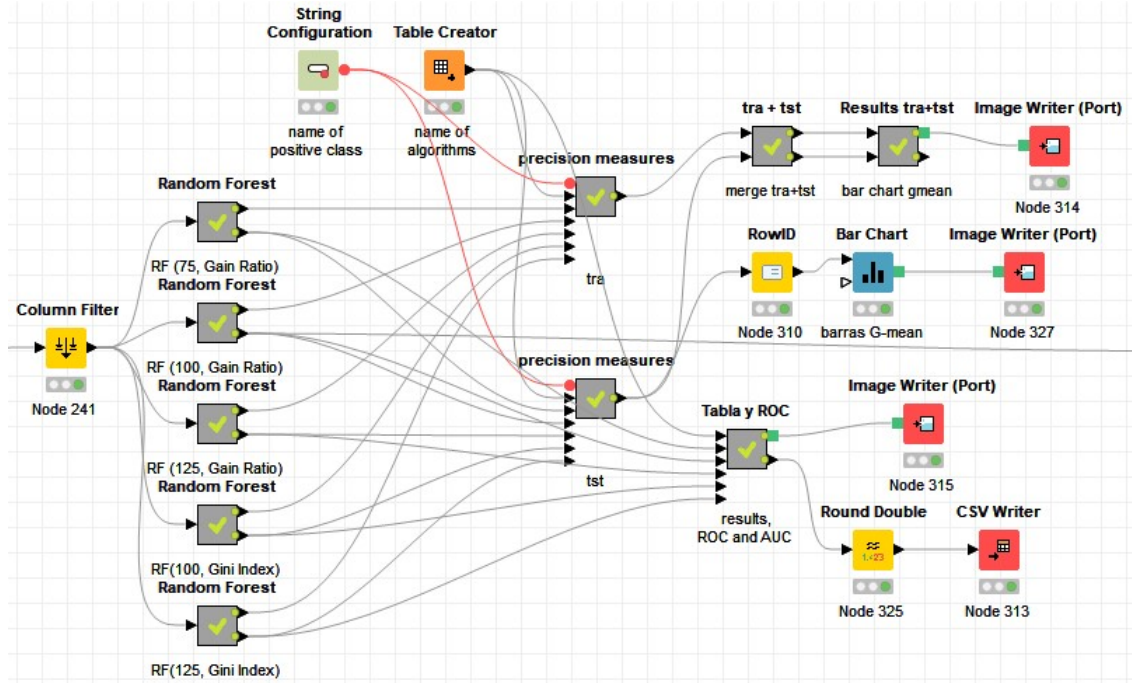


Figura 23: Workflow Random Forest.

Mostraremos el contenido de uno de los nodos de **Random Forest** y cómo hemos configurado cada uno de ellos. Para la configuración de train-test simplemente se ha añadido un nodo Predictor extra que predice las clases con los mismos datos de entrenamiento.

Para la modificación de los parámetros, se ha accedido la configuración del nodo **Random Forest Learner** y se ha modificado la información relacionada con el criterio de división (Gini index o Information Gain ratio) y el número de modelos. Por defecto, el número de modelos es 100 y el criterio de división Gain ratio. También he usado mi número de DNI para inicializar la semilla aleatoria.

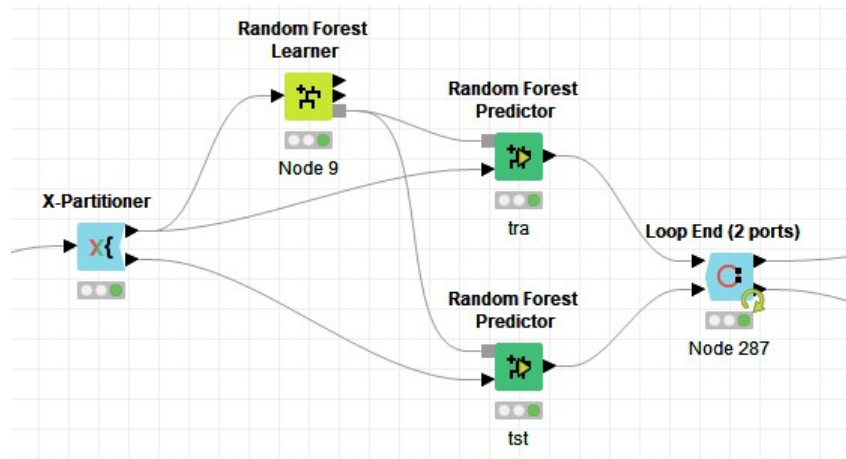


Figura 24: Interior Nodo Random Forest.

The screenshot shows the configuration window for a Random Forest model. It is divided into two main sections: 'Tree Options' and 'Forest Options'. In the 'Tree Options' section, the 'Split Criterion' is set to 'Information Gain Ratio' (indicated by a red arrow). Below this, there are checkboxes for 'Limit number of levels (tree depth)' and 'Minimum child node size', both of which are currently unchecked. In the 'Forest Options' section, the 'Number of models' is set to 100 (indicated by a red arrow). At the bottom, there is a checkbox for 'Use static random seed' which is checked, with a text field containing the seed value '76592621' and a 'New' button.

Figura 25: Configuración Random Forest.

Crearemos 5 metanodos y obtendremos información para:

- Random Forest con criterio Gain Ratio y 75 modelos.
- Random Forest con criterio Gain Ratio y 100 modelos.
- Random Forest con criterio Gain Ratio y 125 modelos.
- Random Forest con criterio Gini Index y 100 modelos.
- Random Forest con criterio Gini Index y 125 modelos.

Extraeremos una tabla para valorar los resultados y dibujaremos la curva ROC, para valorar como mejora o empeora el rendimiento del clasificador para las modificaciones realizadas.

Name	TP	FP	TN	FN	PPV	TPR	TNR	F1-score	Accuracy	G-mean	AUC	OF
(75,GR)	17167	2831	33745	5657	0.858	0.752	0.923	0.802	0.857	0.833	0.919	1.114
(100,GR)	17190	2796	33780	5634	0.86	0.753	0.924	0.803	0.858	0.834	0.92	1.113
(125,GR)	17171	2783	33793	5653	0.861	0.752	0.924	0.803	0.858	0.834	0.92	1.114
(100,GI)	17335	2804	33772	5489	0.861	0.76	0.923	0.807	0.86	0.837	0.922	1.114
(125,GI)	17344	2809	33767	5480	0.861	0.76	0.923	0.807	0.86	0.838	0.923	1.115

Tabla 15: Tabla comparativa modificaciones Random Forest.

En la Gráfica 26 podemos observar como las variaciones en el valor del Accuracy en este caso, son incluso más leves que en el algoritmo anterior siendo prácticamente idénticas. Fijando el número de modelos y variando el criterio de división, la mejora es de una centésima en AUC y de dos en el valor del Accuracy.

A continuación mostraremos el cálculo de G-mean sobre los conjuntos de train-test para valorar el sobreaprendizaje de cada modificación y ver cómo se comporta el algoritmo frente a esto.

En la Gráfica 27, podemos ver como las cinco configuraciones tienen un valor similar de overfitting. El que tiene un valor más cercano a 1 es el de 100 modelos con criterio Gain Ratio. Aún así, todas las modificaciones tienen un valor de G-mean del conjunto de train mayor que el de test.

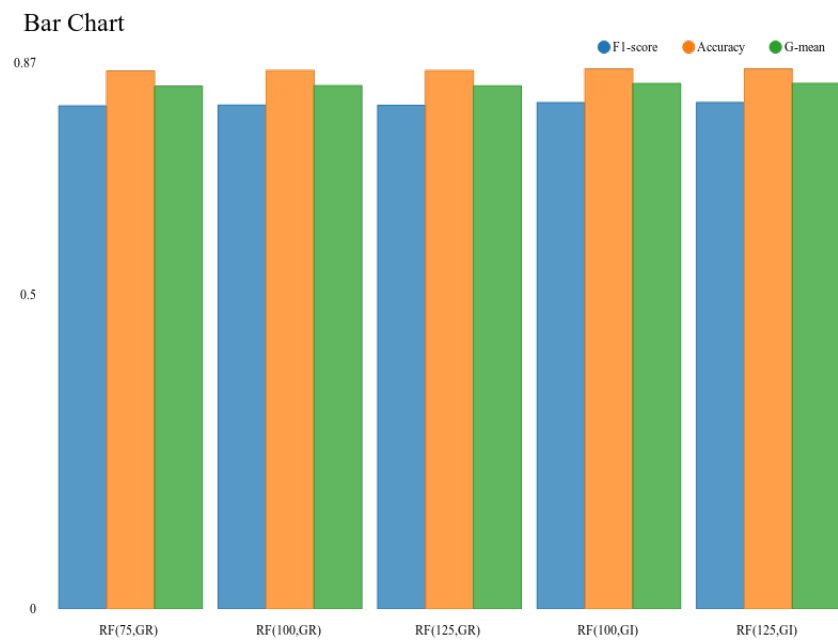


Figura 26: Gráfico de barras medidas Random Forest.

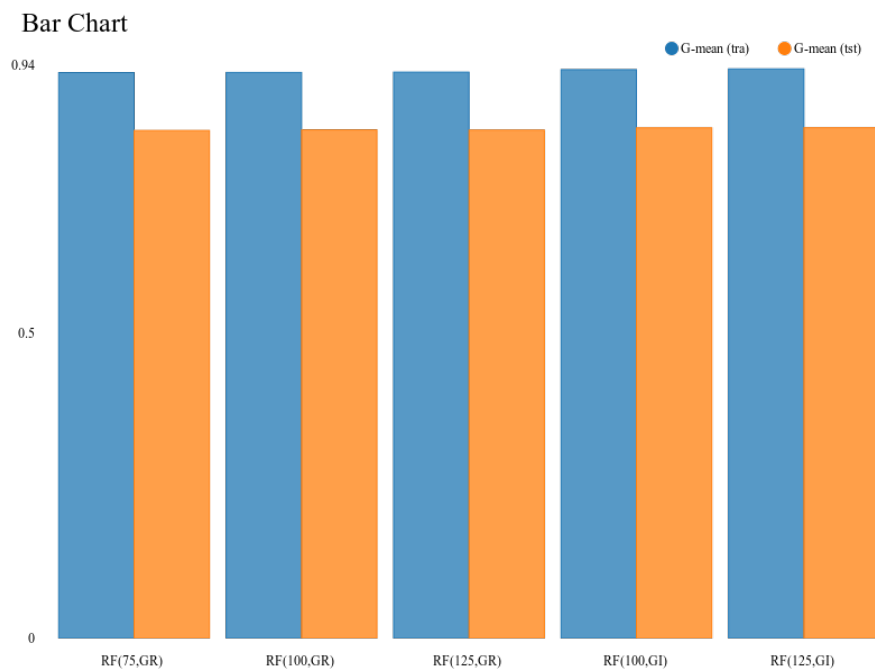


Figura 27: Gráfico de barras G-mean (train-test) Random Forest.

La Gráfica 28 muestra la curva ROC de los 5 casos y veremos qué modificación da un mejor rendimiento. Podemos ver cómo usando el criterio Gain Ratio, podríamos prever que aumentando el número de modelos, el valor del Accuracy y AUC se mantienen, incluso podrían llegar a empeorar si aumentamos la complejidad del modelo. Por otro lado, el criterio Gini index, produce mejores (por algunas centésimas), resultados de Accuracy, AUC y G-mean, pero tienen tendencia a aumentar poco a poco el sobreentrenamiento.

Podemos concluir que a la hora de tener que elegir un algoritmo, me quedaría con la segunda modificación, que tiene un número medio de modelos (100) y obtiene el overfitting más cercano a 1 de las 5 modificaciones.

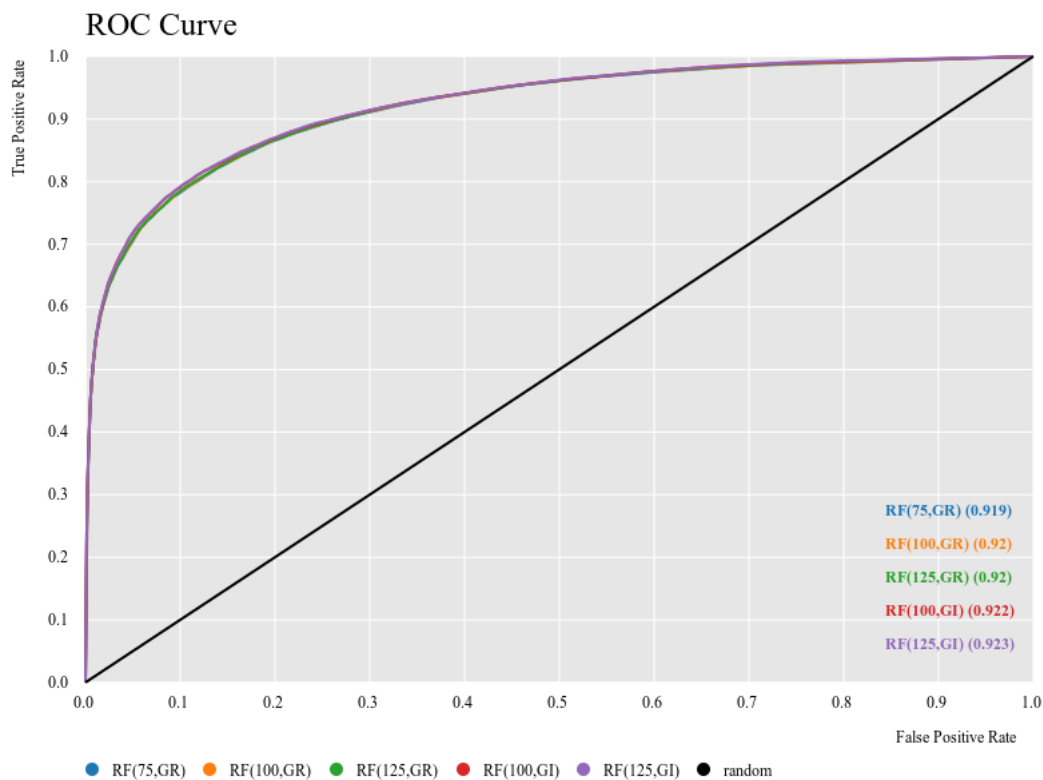


Figura 28: Curva ROC modificaciones Random Forest.

4.3. K-NN.

Para el **K-NN** (K-Nearest Neighbor) hemos realizado 4 modificaciones además de la utilizada por defecto en la comparación inicial. Además en cada caso realizamos un estudio del sobreaprendizaje del mismo.

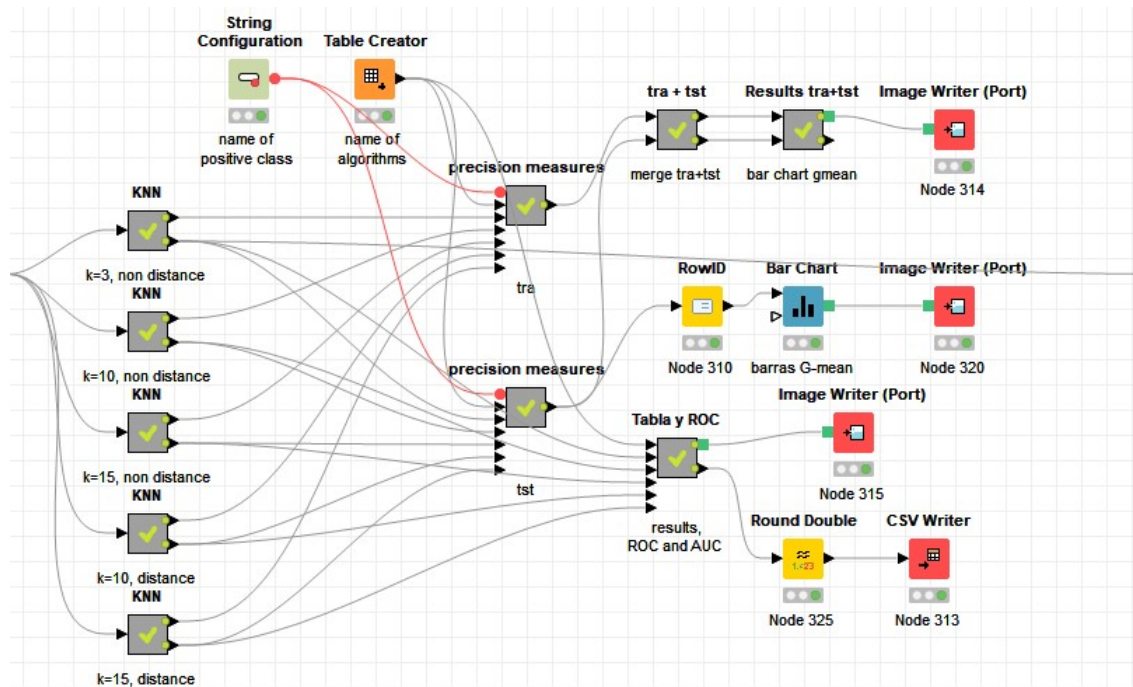


Figura 29: Workflow K-NN.

Mostraremos el contenido de uno de los nodos de **K-NN** y cómo hemos configurado cada uno de ellos. Para la configuración de train-test utilizamos dos nodos **K Nearest Neighbor** que como explicamos antes, cuenta con dos puertos (uno para train y otro para test), en este caso, el nodo para test se configurará de la misma forma, mientras que el de train obtendrá por ambos puertos el conjunto de datos de entrenamiento.

Para la modificación de los parámetros, se ha accedido la configuración del nodo **K Nearest Neighbor** y se ha modificado la información relacionada con el número de vecinos considerados, es decir, el valor de k en el algoritmo y si queremos o no usar la distancia entre los vecinos como peso.

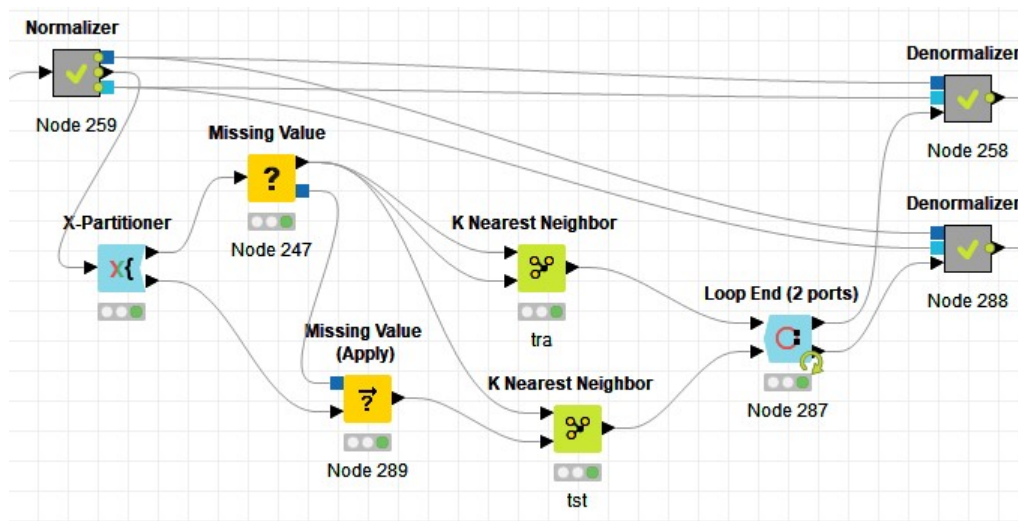


Figura 30: Interior Nodo K-NN.

Column with class labels

S

class

▼

Number of neighbours to consider (k)

3

▲

▼

Weight neighbours by distance

☐

Output class probabilities

☒

Figura 31: Configuración K Nearest Neighbor.

Crearemos 5 metanodos y obtendremos información para:

- K-NN con k=3 y sin considerar la distancia como peso.
- K-NN con k=5 y sin considerar la distancia como peso.
- K-NN con k=10 y sin considerar la distancia como peso.
- K-NN con k=15 y sin considerar la distancia como peso.
- K-NN con k=10 y considerando la distancia como peso.

Antes de mostrar las gráficas hay que tener en cuenta, que se han empezado las modificaciones a partir de k=3, porque era el caso por defecto y porque tomando k=1, como el nodo **K Nearest Neighbor** no implementa Leave One Out; podría dar resultados erróneos, similares a los que se producen al marcar como veremos, la casilla de considerar la distancia como pesos. Por ello, la última modificación ha sido descartada del estudio del overfitting.

Extraeremos una tabla para valorar los resultados y dibujaremos la curva ROC, para valorar como mejora o empeora el rendimiento del clasificador para las modificaciones realizadas.

Name	TP	FP	TN	FN	PPV	TPR	TNR	F1-score	Accuracy	G-mean	AUC	OF
(3,non)	15940	5101	31475	6884	0.758	0.698	0.861	0.727	0.798	0.775	0.841	1.131
(5,non)	15774	5164	31412	7050	0.753	0.691	0.859	0.721	0.794	0.77	0.854	1.092
(10,non)	14866	4608	31968	7958	0.763	0.651	0.874	0.703	0.788	0.755	0.858	1.055
(15,non)	15045	5154	31422	7779	0.745	0.659	0.859	0.699	0.782	0.753	0.854	1.045
(10,dist)	16216	5147	31429	6608	0.759	0.71	0.859	0.734	0.802	0.781	0.871	-

Tabla 16: Tabla comparativa modificaciones K-NN.

En la Gráfica 32 podemos observar la mejor opción de las 5 es la que implementa k=10 y toma la distancia como peso de las características, pero como ya hemos dicho, estos valores son los de test, por lo que no podemos comprobar si esta modificación sobreentrena los datos, así que por falta de conocimiento, no la trataremos como una opción acertada. A continuación, mostraremos el cálculo de G-mean sobre los conjuntos de train-test para valorar el sobreaprendizaje de cada modificación y ver cómo se comporta el algoritmo frente a esto.

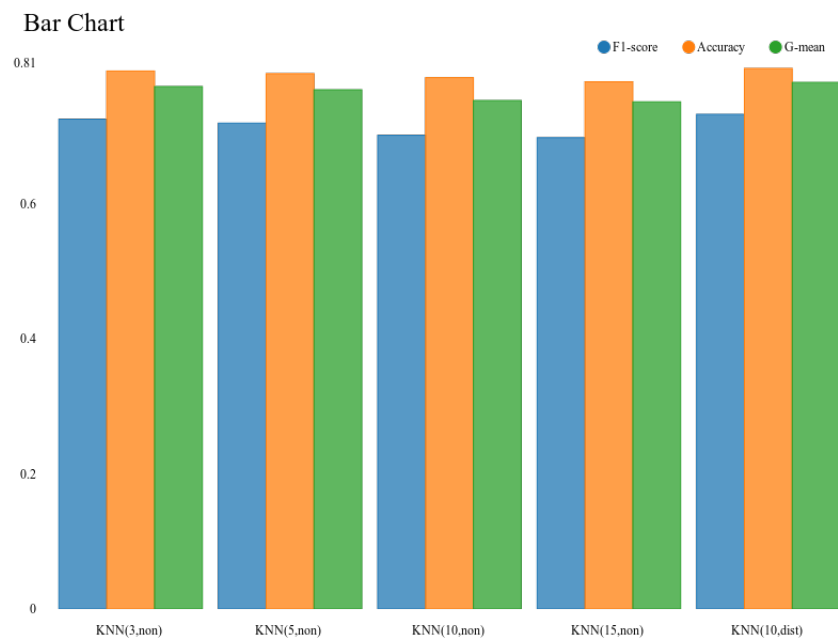


Figura 32: Gráfico de barras medidas K-NN.

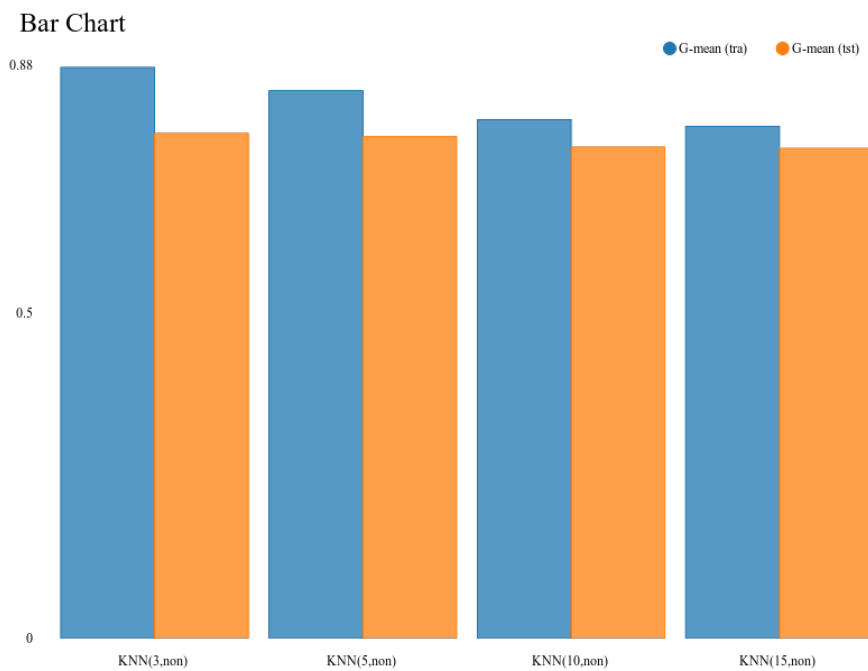


Figura 33: Gráfico de barras G-mean (train-test) K-NN.

En la Gráfica 33, podemos ver como las cuatro configuraciones tienen un valor similar de overfitting. El que tiene un valor más cercano a 1 es la modificación con $k=15$, por lo que podemos ver que aumentando el valor de k , obtenemos un mejor resultado de overfitting.

Finalmente en la Gráfica 34 mostraremos la curva ROC de los 5 casos y veremos qué modificación da un mejor rendimiento. La que mejor rendimiento obtiene es la última modificación, pero al no poder calcular de forma fiel su valor de overfitting, elegiremos de entre los 4 restantes el que ofrezca en media mejores resultados.

En este caso, es evidente que aunque aumentando k , mejoremos el valor del overfitting, el valor de AUC, Accuracy, F1-score y G-mean disminuyen con $k=15$ respecto a su modificación anterior. Esto puede hacernos ver que aumentar el tamaño del entorno para este conjunto de datos puede que no sea beneficioso, por lo que para un valor entre $k=10$ y $k=15$, se obtendrá el mejor rendimiento, fijando el resto de condiciones de la configuración tal y como las tenemos. Por tanto en este caso me quedaría con la tercera modificación, K-NN con $k=10$.

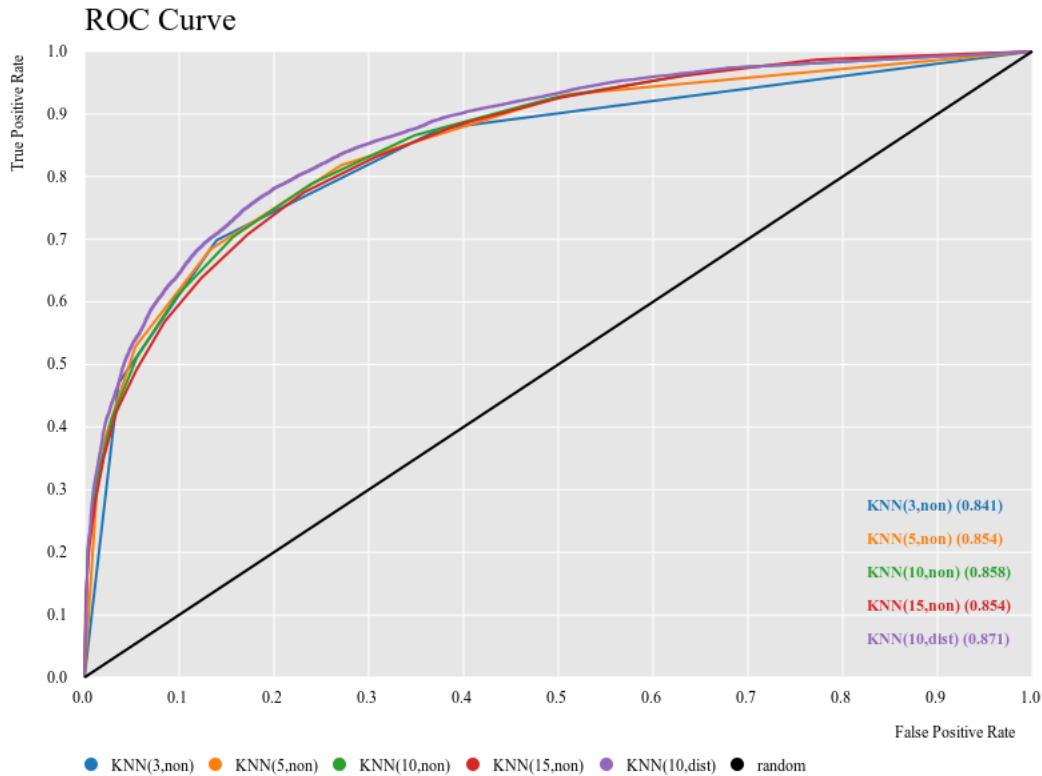


Figura 34: Curva ROC modificaciones K-NN.

Finalmente, se ha realizado el cálculo del overfitting para el resto de algoritmos y sus casos por defecto. El entorno de trabajo de KNIME ha sido común para todos a los que se ha agregado un nodo Predictor extra tal y como se ha hecho en los casos de **C4.5** y **Random Forest**.

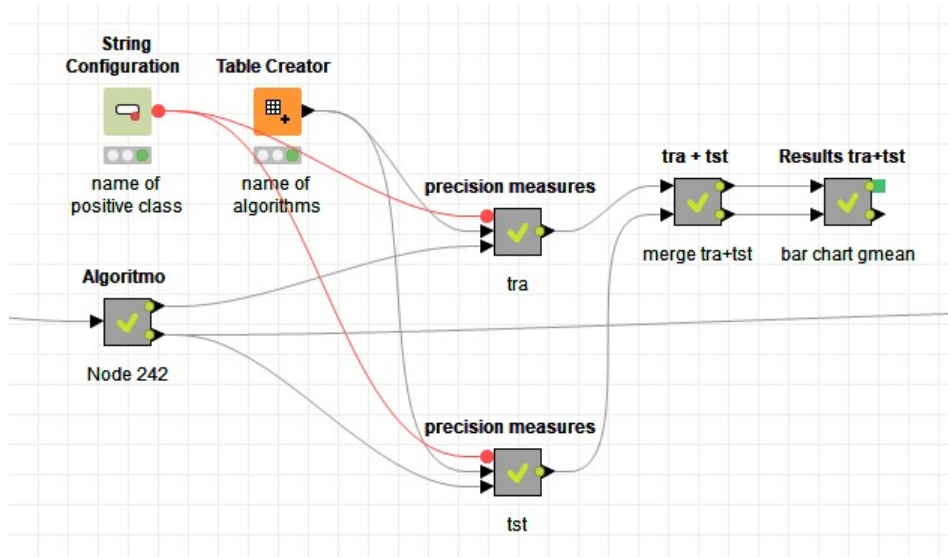


Figura 35: Workflow Algoritmo train-test.

Se mostrará una tabla comparativa de algunas medidas para train y test. No se adjuntará la tabla completa debido a sus dimensiones y a que no aporta más información que la que se precisará a continuación.

Name	F1-score(tra)	F1-score(tst)	Accuracy(tra)	Accuracy(tst)	G-mean(tra)	G-mean(tst)	OF
C4.5	0.935	0.771	0.95	0.826	0.946	0.813	1.164
Gradient Boosted	0.763	0.75	0.833	0.825	0.8	0.791	1.012
Naive Bayes	0.655	0.655	0.759	0.759	0.716	0.716	1.001
Random Forest	0.919	0.802	0.94	0.857	0.928	0.833	1.114
KNN	0.853	0.727	0.891	0.798	0.877	0.775	1.131
Neural Network	0.656	0.652	0.753	0.751	0.717	0.714	1.004

Tabla 17: Tabla comparativa train-test.

También mostramos el gráfico de barras de G-mean (train) y G-mean (test), para hacer más visual la comparación del overfitting de los diferentes algoritmos.

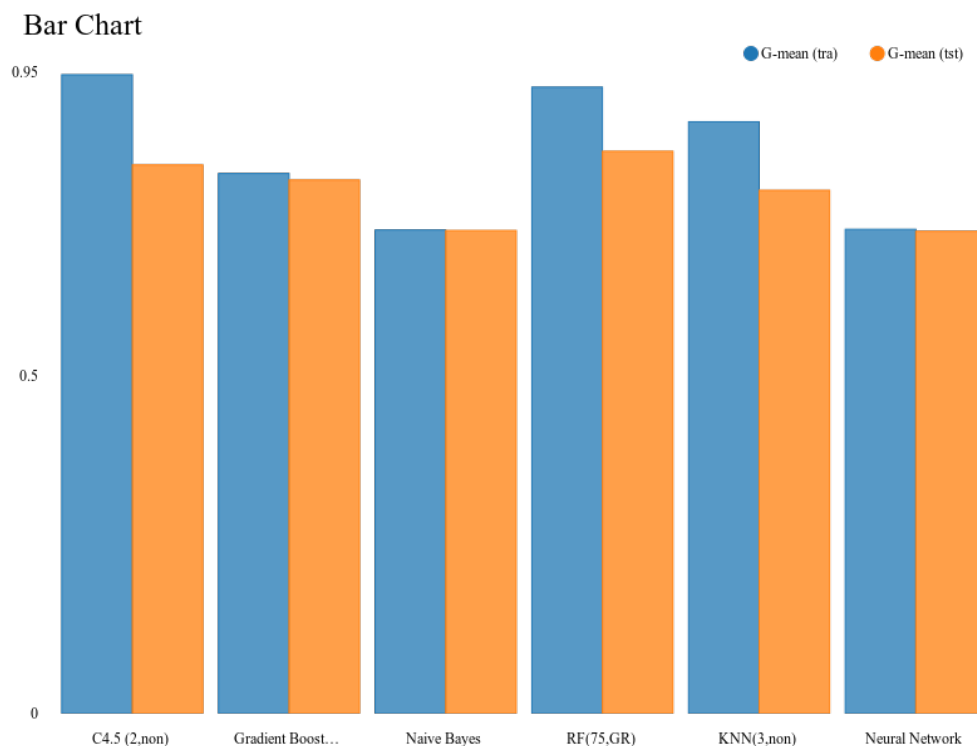


Figura 36: Gráfica de barras overfitting.

En la Gráfica 36 podemos observar, como **Naive Bayes** y **Neural Network**, a pesar de ser los que peores resultados obtienen de forma general en los valores de test, también son los que tienen un overfitting más equilibrado. Como hemos visto en teoría, los clasificadores que emplean árboles de decisión son los más propensos a problemas de sobreaprendizaje. Aunque también vimos que era uno de los problemas de las redes neuronales y en este caso, no ocurre, probablemente y como ya dijimos en el segundo apartado, una mayor cantidad de datos haría que obtuviésemos mejores resultados en la red neuronal aumentando el rendimiento pero también el sobreentrenamiento ya que las redes neuronales se basan en gran parte en ensayo y error.

Por otro lado, el algoritmo **C4.5** en su configuración por defecto es el que mayor sobreentrenamiento produce, aunque como vimos en clase y como hemos visto al modificar su configuración, podemos arreglarlo con la poda.

5. Procesado de datos.

Para el procesado de los datos he realizado dos modificaciones sobre el dataset inicial. Primeramente es evidente que no existía un balance total de las tres clases del problema. Utilizando el nodo **Data Explorer**, podemos observar el desbalance de los ejemplos de cada clase.



Figura 37: Distribución de las 3 clases del problema.

Por tanto, realizaremos un muestreo de los datos del conjunto con el objetivo de equilibrar los ejemplos de cada clase e igualar las 3 clases al número de muestras de la clase intermedia (*non functional*). Para ello, usaremos los nodos **Equal Size Sampling** y **Bootstrap Sampling**.

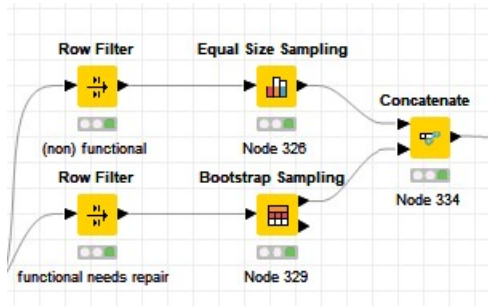


Figura 38: Balance de las muestras del problema.

Primero dividiremos el conjunto de datos en dos subconjuntos, uno que contenga los ejemplos de las clases *functional* y *non functional* y otro con los de la clase *functional needs repair*. Una vez hecho esto, utilizando **Equal Size Sampling** igualaremos el tamaño de los ejemplos de *functional* y *non functional*, internamente se hará un under-sampling al conjunto *functional* hasta igualarlo a las 22824 muestras del *non functional*.

Por otro lado, se realizará un oversampling sobre el conjunto de datos de la clase *functional needs repair* con el nodo **Bootstrap Sampling** que añade nuevos ejemplos

generados por repetición aleatoria de muestras existentes hasta alcanzar el tamaño especificado. Finalmente concatenando los resultados, obtendremos el nuevo dataset con las tres clases balanceadas como podemos observar con **Data Explorer**.

class	<input type="checkbox"/>	0	3	functional, functional needs repair, non functional	22824
-------	--------------------------	---	---	---	-------

Figura 39: Nueva distribución de las 3 clases del problema.

El balance de las 3 clases nos proporcionará como resultado una clasificación más equilibrada, además aumentará el rendimiento de los algoritmos y reducirá el sobreentrenamiento sobre un conjunto de muestras de la misma clase.

La segunda modificación ha sido filtrar las columnas que tuviesen una correlación lineal de 1 con otras, es decir, columnas que ofrecen información redundante que ya proporcionan otras o que directamente están repetidas.

Para ello, utilizaré el nodo **Linear Correlation** y me ayudaré del nodo **Correlation Filter** para no tener que eliminar las columnas de forma manual. Este procesado, eliminará el ruido del conjunto de datos y aumentará la rapidez y el rendimiento de los algoritmos.

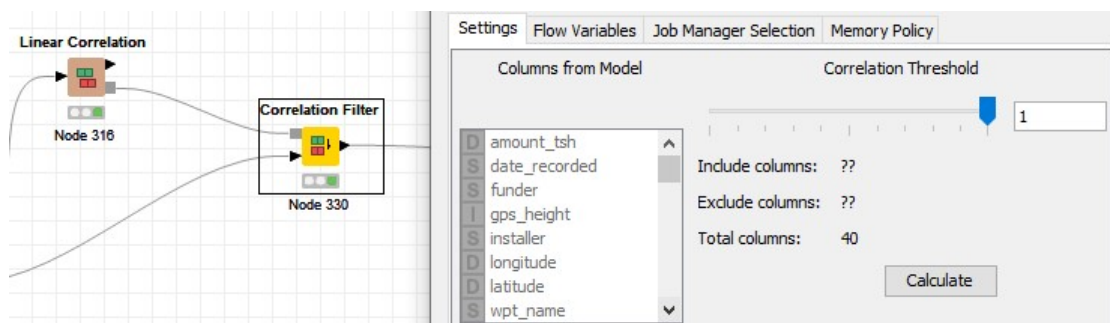


Figura 40: Configuración Correlation Filter.

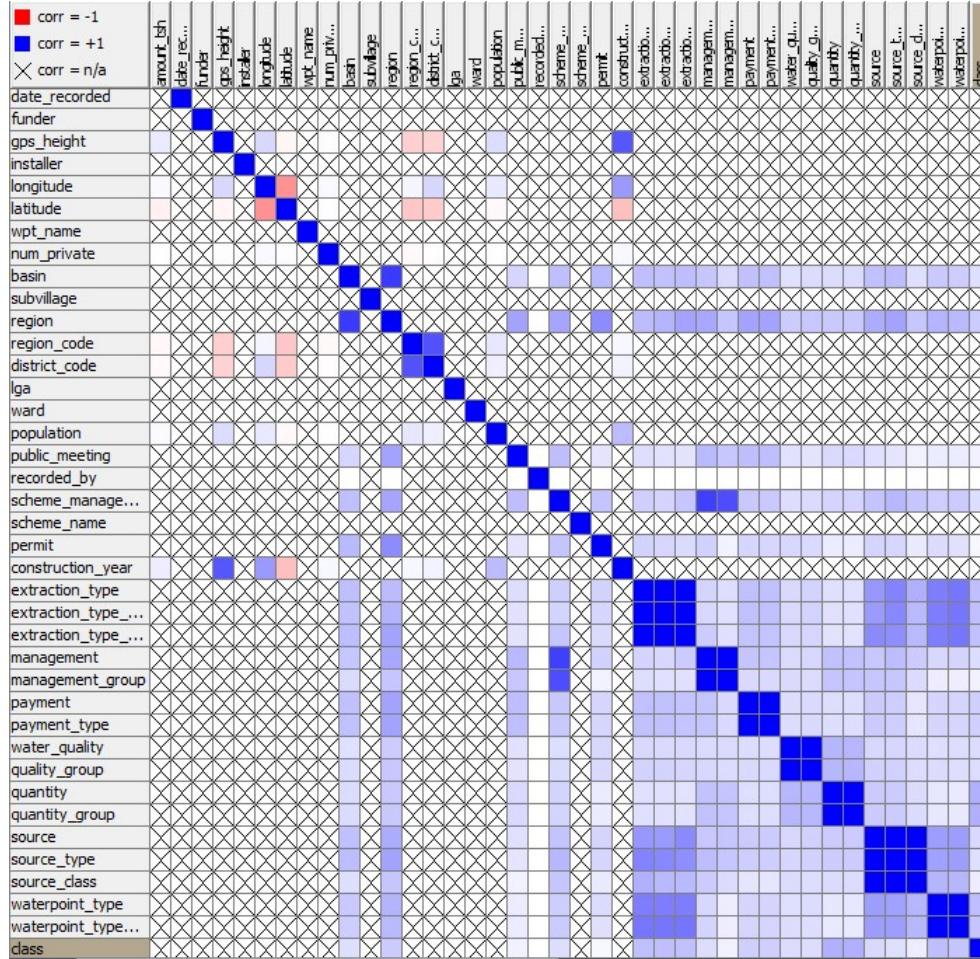


Figura 41: Correlación Lineal entre las distintas categorías.

Con este procesado se eliminan un total de 7 columnas, podríamos bajar el coeficiente de correlación a un valor más bajo y quizás eliminar alguna columna más, pero he decidido que sólo voy a eliminar las columnas si existe una correlación de 1, ya que de la otra forma podríamos eliminar información importante o perjudicar a algunos algoritmos que le saquen ventaja a la misma.

Una vez realizadas estas dos modificaciones, utilizaremos el nuevo dataset para ejecutar de nuevo todos los algoritmos y modificaciones de los mismos que ya hemos tratado en los diferentes apartados. Para no volver a mostrar la información detallada de cada algoritmo, mostraré las tablas del antes y después y me limitaré a comparar la mejora o empeoramiento de los algoritmos.

Mostraremos a continuación la tabla común de los 6 algoritmos antes y después del procesamiento y veremos como afecta a cada uno y la curva ROC de los nuevos algoritmos v2.0.

Name	Accuracy	AUC	F1-score	FN	FP	G-mean	PPV	TN	TNR	TP	TPR	OF
C4.5	0.826	0.841	0.771	5373	4971	0.813	0.778	31605	0.864	17451	0.765	1.164
C4.5 2.0	0.868	0.877	0.795	5303	3724	0.84	0.825	41924	0.918	17521	0.768	1.131
Gradient Boosted	0.825	0.889	0.75	7191	3206	0.791	0.83	33370	0.912	15633	0.685	1.012
Gradient Boosted 2.0	0.836	0.886	0.733	7422	3813	0.786	0.802	41835	0.916	15402	0.675	1.01
Naive Bayes	0.759	0.8	0.655	9248	5085	0.716	0.728	31491	0.861	13576	0.595	1.001
Naive Bayes 2.0	0.748	0.793	0.617	8919	8350	0.706	0.625	37298	0.817	13905	0.609	1.001
Random Forest	0.858	0.92	0.803	5634	2796	0.834	0.86	33780	0.924	17190	0.753	1.114
Random Forest 2.0	0.887	0.942	0.817	5477	2280	0.85	0.884	43368	0.95	17347	0.76	1.095
KNN	0.798	0.841	0.727	6884	5101	0.775	0.758	31475	0.861	15940	0.698	1.131
KNN 2.0	0.83	0.884	0.731	7052	4582	0.788	0.775	41066	0.9	15772	0.691	1.111
MLP	0.751	0.799	0.652	8952	5859	0.714	0.703	30717	0.84	13872	0.608	1.004
MLP 2.0	0.753	0.813	0.636	8085	8798	0.722	0.626	36850	0.807	14739	0.646	1.001

Tabla 18: Tabla comparativa para los algoritmos (antes-después).

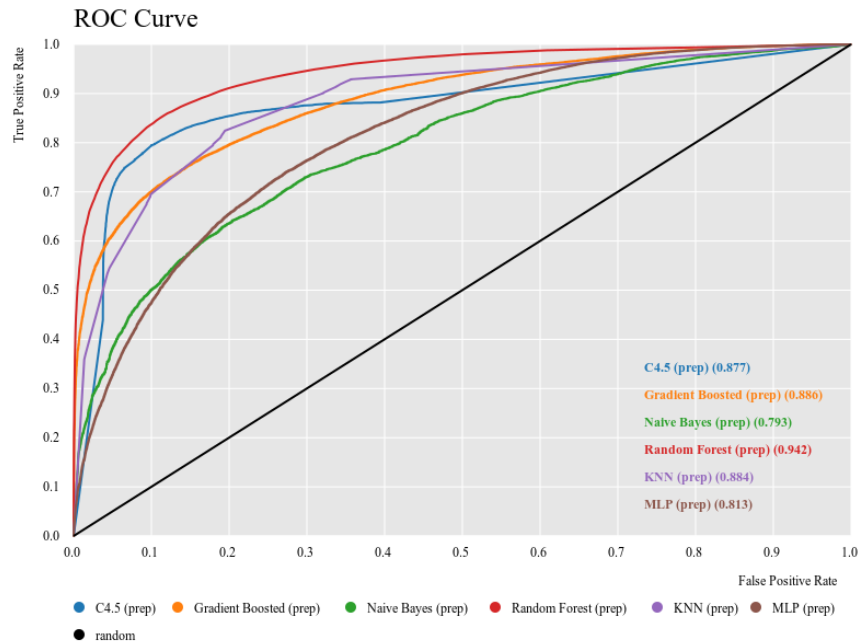


Figura 42: Curva ROC algoritmos v2.0.

En la Tabla 18 podemos observar como todos los algoritmos menos **Naive Bayes**, han mejorado su Accuracy, es decir, han aumentado en mayor o menor medida su tasa de acierto respecto a la versión anterior.

Naive Bayes probablemente empeora su rendimiento debido a que como ya dijimos, supone que las variables del problema, son independientes, por lo que la modificación de la correlación lineal no ha supuesto ninguna mejora, es más, la ausencia de menos columnas, ha empeorado su tasa de clasificación.

Por otra parte, el algoritmo que se ha visto más beneficiado, ha sido **C4.5**, seguido muy de cerca por **Random Forest** y **KNN**. Estos dos últimos los analizaremos en más detalle a continuación. La nueva versión de **C4.5** ha disminuido el valor de FP y ha aumentado los valores de F1-score y G-mean, obteniendo en media mejores resultados que su versión anterior. Además ha reducido su índice de overfitting, lo que hace al caso por defecto de **C4.5** mucho mejor que antes.

A continuación veremos las mejoras que han supuesto a las modificaciones del apartado 4 sobre el nuevo conjunto de datos para 2 algoritmos: **Random Forest** y **K-NN**.

5.1. Random Forest.

Mostraremos la tabla comparativa para las modificaciones de Random Forest antes y después del procesado.

Name	TP	FP	TN	FN	PPV	TPR	TNR	F1-score	Accuracy	G-mean	AUC	OF
(75,GR)	17167	2831	33745	5657	0.858	0.752	0.923	0.802	0.857	0.833	0.919	1.114
(75,GR) 2.0	17345	2322	43326	5479	0.882	0.76	0.949	0.816	0.886	0.849	0.942	1.097
(100,GR)	17190	2796	33780	5634	0.86	0.753	0.924	0.803	0.858	0.834	0.92	1.113
(100,GR) 2.0	17347	2280	43368	5477	0.884	0.76	0.95	0.817	0.887	0.85	0.942	1.094
(125,GR)	17171	2783	33793	5653	0.861	0.752	0.924	0.803	0.858	0.834	0.92	1.114
(125,GR) 2.0	17375	2299	43349	5449	0.883	0.761	0.95	0.818	0.887	0.85	0.943	1.095
(100,GI)	17335	2804	33772	5489	0.861	0.76	0.923	0.807	0.86	0.837	0.922	1.114
(100,GI) 2.0	17651	2405	43243	5173	0.88	0.773	0.947	0.823	0.889	0.856	0.943	1.095
(125,GI)	17344	2809	33767	5480	0.861	0.76	0.923	0.807	0.86	0.838	0.923	1.115
(125,GI) 2.0	17677	2428	43220	5147	0.879	0.774	0.947	0.824	0.889	0.856	0.943	1.095

Tabla 19: Tabla comparativa modificaciones Random Forest (antes-después).

En rasgos generales, el nuevo conjunto dataset produce mejores resultados para todas las modificaciones, obteniendo el que ha sido por ahora el mejor valor de AUC de todos los algoritmos y modificaciones vistas (0.943). Todos los casos han reducido su tasa de overfitting, por lo que podemos ver que las modificaciones realizadas mejoran el problema del sobreaprendizaje.

Las 3 últimas modificaciones ofrecen unos resultados similares por lo que nos quedaremos con la que menor complejidad aporta al modelo, el caso con 100 modelos y criterio Gini index.

A continuación mostraremos la curva ROC de las nuevas modificaciones:

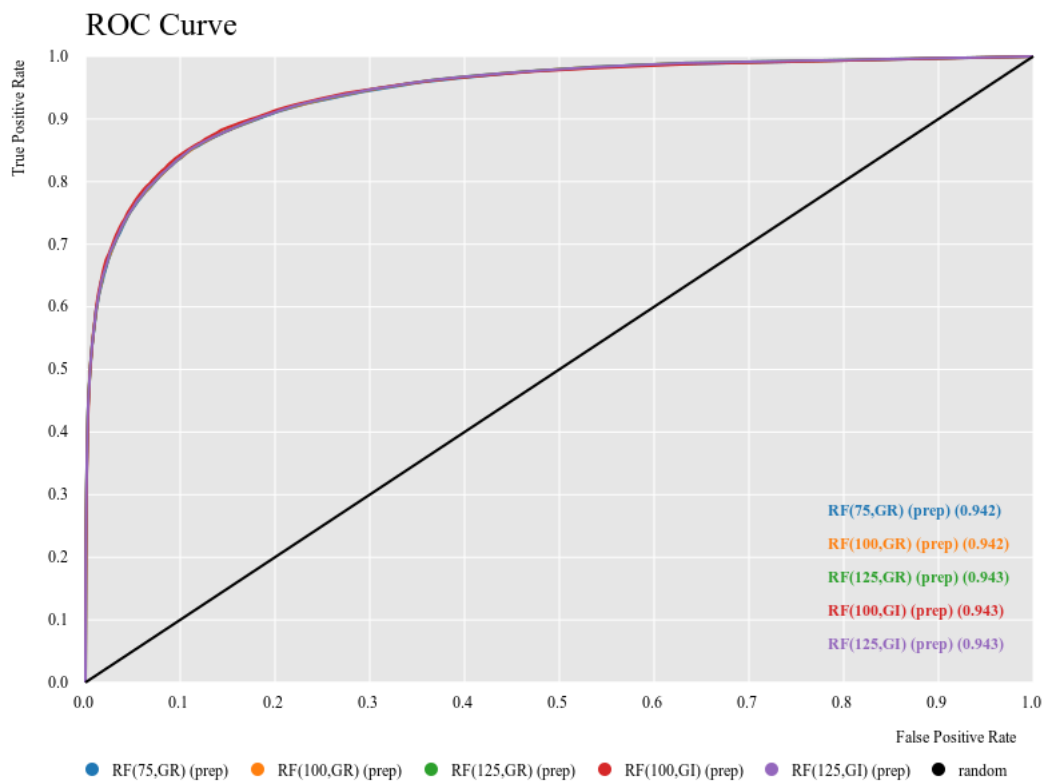


Figura 43: Curva ROC Random Forest v2.0.

5.2. K-NN.

En este caso, además de la comparación de los resultados de la primera versión con los de la nueva, añadiremos un extra basado en el procesado de datos con el nodo **One to Many** que convierte los diferentes valores nominales de cada categoría, en nuevas categorías que indican con 1 o 0 si presentan o no ese valor.

Esta modificación, la haré para **K-NN** puesto que es evidente que si trabajamos con nuevas categorías y además numéricas, al trabajar con distancias, podría mejorar los resultados anteriores. Además esta modificación la haré con las categorías que tengan menos de 20 valores, basándome en el nodo **Data Explorer**, para evitar añadir más columnas de la cuenta y obtener un peor rendimiento.

Las 14 categorías seleccionadas (todas ellas con un rango de atributos inferior a 20) para convertir sus atributos en categorías para la v3.0 han sido:

- basin (String)
- public_meeting (String)
- recorded_by (String)
- scheme_management (String)
- permit (String)
- extraction_type (String)
- management (String)
- management_group (String)
- payment (String)
- water_quality (String)
- quality_group (String)
- quantity (String)
- source (String)
- waterpoint_type (String)

Esta nueva versión tendrá 123 columnas de las cuales 90 serán de ceros o unos, ya que se han añadido a partir del procesado del nodo **One to Many**, eso obviamente hará más lenta la ejecución del algoritmo **K-NN**, así que veremos si los resultados obtenidos son mucho mejores o por el contrario es preferible sacrificar unas milésimas en la mejora por una mayor velocidad en la ejecución.

Name	TP	FP	TN	FN	PPV	TPR	TNR	F1-score	Accuracy	G-mean	AUC	OF
k=3	15940	5101	31475	6884	0.758	0.698	0.861	0.727	0.798	0.775	0.841	1.131
k=3 2.0	16141	4685	40963	6683	0.775	0.707	0.897	0.74	0.834	0.797	0.875	1.111
k=3 3.0	16885	3751	41897	5939	0.818	0.74	0.918	0.777	0.858	0.824	0.895	1.089
k=5	15774	5164	31412	7050	0.753	0.691	0.859	0.721	0.794	0.77	0.854	1.092
k=5 2.0	15772	4582	41066	7052	0.775	0.691	0.9	0.731	0.83	0.788	0.884	1.079
k=5 3.0	16353	3512	42136	6471	0.823	0.716	0.923	0.766	0.854	0.813	0.903	1.063
k=10	14866	4608	31968	7958	0.763	0.651	0.874	0.703	0.788	0.755	0.858	1.055
k=10 2.0	15485	5077	40571	7339	0.753	0.678	0.889	0.714	0.819	0.777	0.882	1.067
k=10 3.0	16074	3828	41820	6750	0.808	0.704	0.916	0.752	0.846	0.803	0.905	1.048
k=15	15045	5154	31422	7779	0.745	0.659	0.859	0.699	0.782	0.753	0.854	1.045
k=15 2.0	15395	5570	40078	7429	0.734	0.675	0.878	0.703	0.81	0.77	0.878	1.042
k=15 3.0	15815	4101	41547	7009	0.794	0.693	0.91	0.74	0.838	0.794	0.901	1.033

Tabla 20: Tabla comparativa modificaciones K-NN (antes-después).

A la vista de los resultados, hemos mejorado cada versión con cada mejora, obteniendo en la v3.0, que incluye los 3 procesados, los mejores resultados. Con cada procesado, mejoramos el overfitting, así como el Accuracy, G-mean y AUC. Para el caso k=15 ocurre en la v2.0 y v3.0 lo mismo que ya ocurría en la versión inicial, empeora los resultados de la versión k=10 con sus respectivas mejoras, por lo que sería recomendable quedarnos con la v3.0 con k=10.

Finalmente y como ya preeveíamos, el procesado del nodo **One to Many** proporciona mejores resultados sobre cada parámetro debido a que al categorizar los atributos y darles un valor 0 o 1, mejoramos dos cosas:

- Evitamos asignar en el **Number to Category** una cantidad ordinal predefinida (1,2,3...) que no tiene sentido para nuestros valores categóricos.
- En el proceso de normalización, el rango está definido (0 o 1). Aportamos información de existencia o no de esa categoría, evitando así valores intermedios que no aportan información útil en este problema.

Finalmente mostraremos las curvas ROC para ambos procesados como hemos hecho con los algoritmos anteriores:

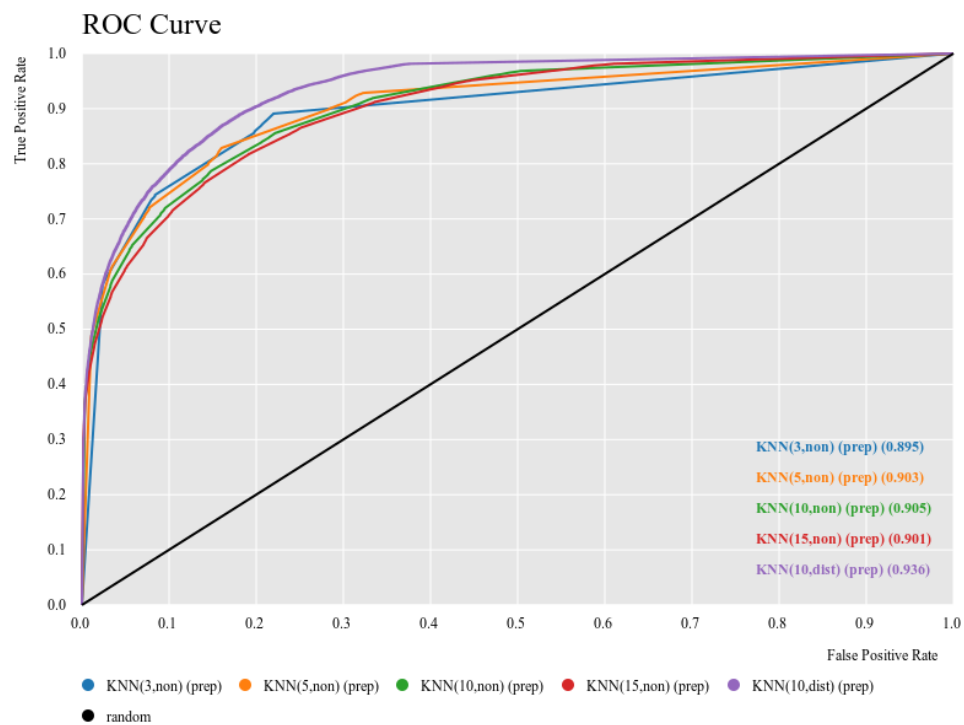
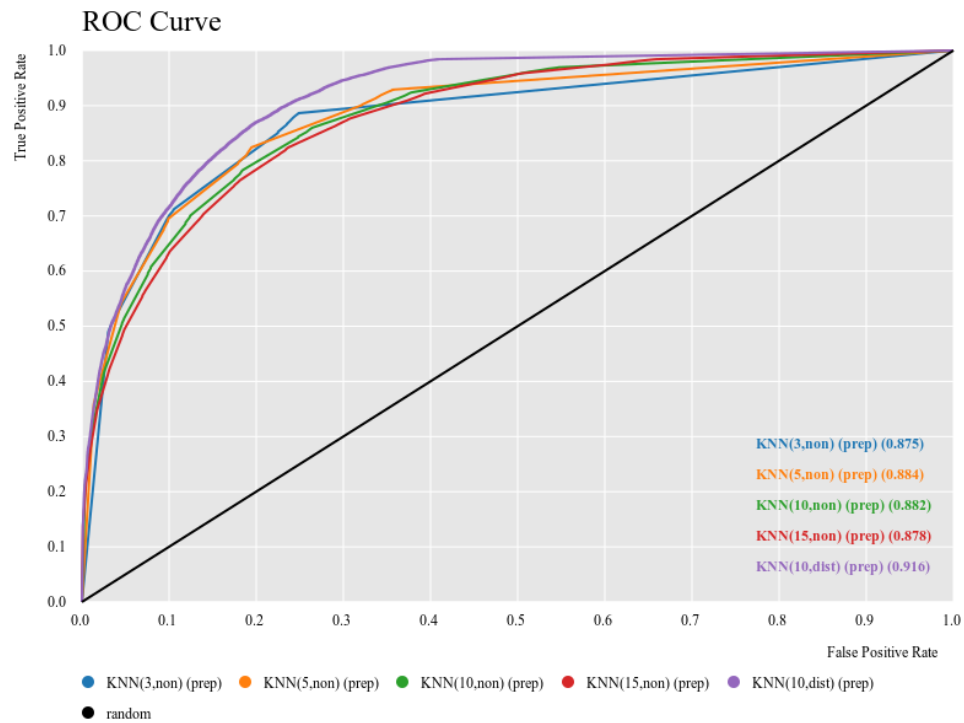


Figura 44: Curva ROC K-NN v2.0 (arriba) y v3.0 (abajo).

6. Interpretacion de resultados.

A continuación mostraremos un gráfico con las medidas más relevantes calculadas y cómo varía dependiendo del algoritmo, los cuales he tomado las versiones de la mejora.

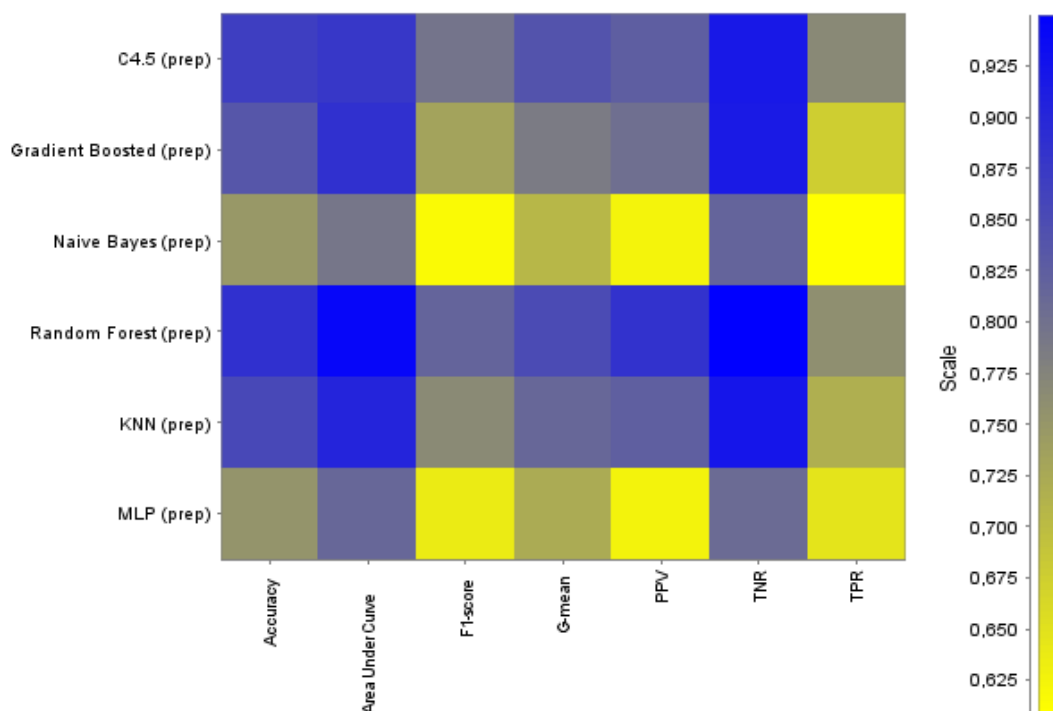


Figura 45: Heat Map de las medidas de los diferentes algoritmos.

Para cada algoritmo cuanto mayor sea la tonalidad azul en su fila, mejor será para el análisis de este conjunto de datos. Como ya hemos analizado antes, el mejor algoritmo es **Random Forest** seguido de **C4.5** y **KNN** con las mejoras realizadas. Por lo que me quedaría con la versión de **Random Forest** mejorada (100 modelos, criterio Gini index).

Utilizando el modelo de árbol de decisión generado en **Decision Tree Learner** en su versión 2.0, analizaremos las características más importantes para determinar la clase de un ejemplo de los datos. A raíz del análisis, **C4.5** ofrece unos buenos resultados de media, así que podrá ofrecernos información medianamente fiable para determinar qué características son las más importantes. Utilizaremos el nodo **Decision Tree View** para exportar la imagen.

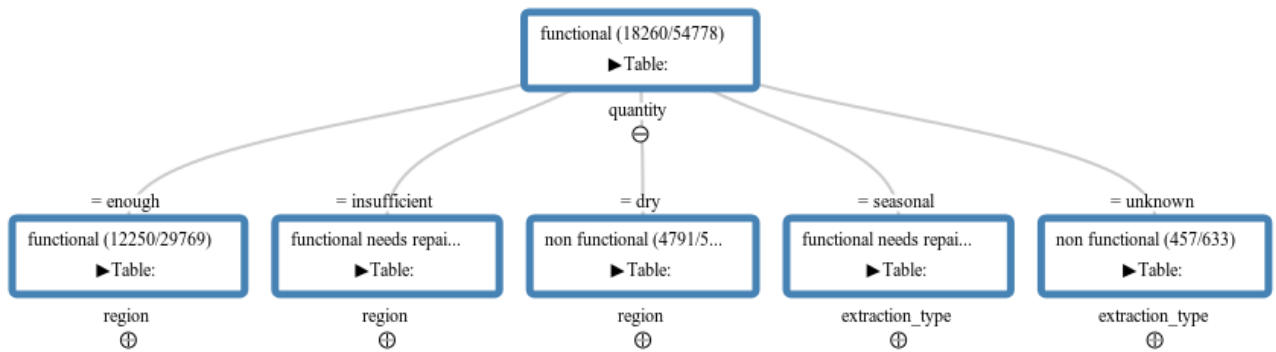


Figura 46: Primer nivel del árbol de decisión C4.5.

La característica usada para poder predecir cual es el estado de la bomba de agua es *quantity*. Es bastante lógico que nos basemos en la cantidad de agua existente debido a que si está seca, probablemente no funcione, si hay suficiente agua funcionará y si hay insuficiente o algunas veces hay y otras no, sufrirá algún problema que habrá que reparar.

Si extraemos las medidas de correlación que nos devuelve el nodo **Linear Correlation** podemos ver con qué característica existe mayor correlación con la clase.

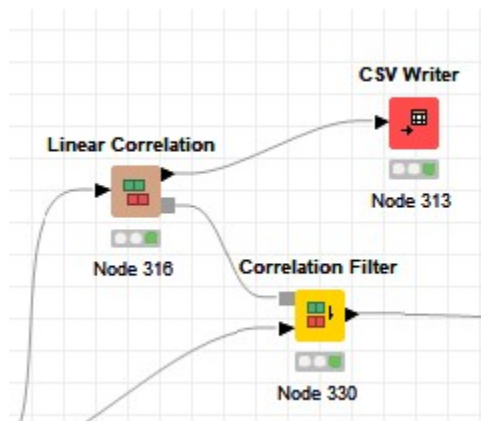


Figura 47: Extracción de las medidas de correlación.

Para no incluir toda la tabla mostraremos el fragmento que nos interesa:

	quantity	extraction_type	region	class
quantity	1	0.1218	0.2138	0.3158
extraction_type	0.1218	1	0.2558	0.2649
region	0.2138	0.2558	1	0.2577
class	0.3158	0.2649	0.2577	1

Tabla 21: Tabla con las medidas de correlación más relevantes.

Podemos ver que el mayor valor se corresponde con *quantity* como ya habíamos dicho observando el modelo del árbol de decisión. Las otras dos categorías que le siguen son la región y el tipo de extracción que también serán relevantes para predecir la clasificación del conjunto de datos.

7. Contenido adicional.

Se consideran adicionales los contenidos que se han desarrollado a lo largo de la práctica y se han usado para complementar la información obtenida.

Procesado extra (Muestreo de clases, One to Many), comparación train-test de los algoritmos...

8. Bibliografía.

[1]. Diapositivas de la asignatura.

[2]. Vídeo-tutoriales sobre la configuración de los algoritmos en KNIME.

- Comparación algoritmos en KNIME: <https://youtu.be/wah7TFY933o>
- Obtención del decision tree size en KNIME: <https://youtu.be/H21otPlycA4>
- Obtención curva ROC y AUC en KNIME: <https://youtu.be/OQnfOFlySMM>
- Análisis del overfitting de train vs test en KNIME: <https://youtu.be/NsgY7KJabOg>