UNIVERSIDAD DE GRANADA

Inteligencia de Negocio

Práctica 1

Análisis predictivo mediante clasificación

David Carrasco Chicharro

davidcch@correo.ugr.es

Índice

1. Introducción

El problema abordado en esta práctica trata de utilizar algoritmos de aprendizaje supervisado de clasificación como herramienta para realizar un análisis predictivo. Se evaluarán los distintos algoritmos utilizados en función de sus resultados ejecutándolos con su configuración por defecto y con ciertos ajustes paramétricos después. Además se realizará un preprocesado de datos para intentar conseguir una mejor solución. Para este problema en concreto se dispone de un conjunto con 59400 instancias con 39 variables cada una. El análisis se realizará para predecir qué bombas de extracción de agua funcionan, cuáles necesitan algunas reparaciones y cuáles no funcionan, centrándonos sobre todo en estas últimas (clase positiva). Los algoritmos de clasificación elegidos para la práctica son de distintos tipos:

- C4.5 (árboles): es uno de los algoritmos basados en árboles de decisión más populares, capaz de manejar tanto variables continuas como discretas. Se basa en la utilización del criterio de proporción de ganancia, de modo que consigue evitar que variables con mayor número de posibles valores salgan beneficiadas en la selección. Proporciona un árbol de decisión como resultado relativamente fácil de entender.

- Naïve-Bayes (probabilístico): al igual que el algoritmo anterior, es bastante conocido en la literatura dentro del conjunto de algoritmos de clasificación probabilísticos. Utiliza métodos bayesianos para estimar cada variable de manera independiente al resto, lo cual es interesante para este problema dada la gran cantidad de variables con las que se cuentan.

- OneR (reglas): es un algoritmo extremadamente sencillo que puede servir como base para determinar que el resto de algoritmos deben superar los resultados de éste.

- K-NN (distancias): dado que el conjunto de instancias no es extremadamente grande, este algoritmo es interesante por su capacidad de generar un modelo en base a sus vecinos más cercanos, de modo que no resulta demasiado costoso evaluar varios k-vecinos y comparar con todas las instancias. Además realiza suposiciones sobre los conceptos relevantes que afectan a la clase final, lo cual es un punto a favor dado el gran conjunto de variables que se presentan.

- Random Forest (multiclasificador bagging): se trata de uno de los algoritmos de clasificación más utilizados y presenta muy buenos resultados, especialmente cuando hay datos suficientes como es el caso de esta práctica. La preparación de datos necesaria es mínima y es capaz de identificar las variables más significativas.
- Grandien Boosting (multiclasificador boosting): es un algoritmo extremadamente popular en machine learning y uno de los métodos líderes entre los ganadores de las competiciones de Kaggle. No requiere gran preprocesamiento de datos y funciona bien tanto con variables categóricas como numéricas, como es nuestro caso práctico. Además maneja valores perdidos sin requerir imputación, lo cual nos resulta favorecedor dado que en nuestro ejemplo hay una gran cantidad de estos.

Una vez elegidos los algoritmos, la primera ejecución de ellos se realizará con la configuración por defecto y sin ningún preprocesado de datos. La siguiente ejecución se realizará con un mínimo preprocesado y con la configuración óptima de cada algoritmo.

2. Resultados obtenidos

2.1. C4.5

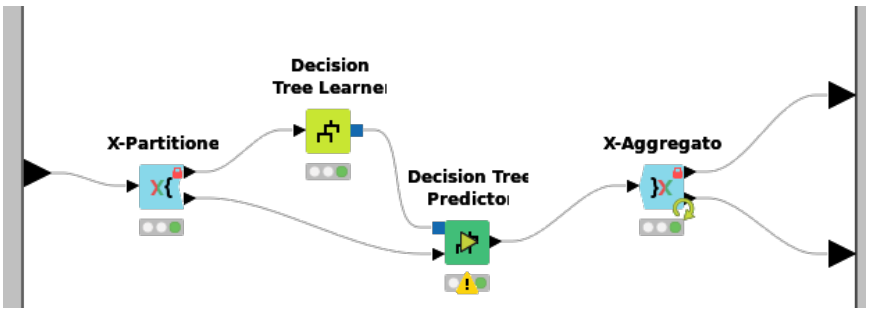

Figura 1: Flujo de trabajo de C4.5

	TP	FP	TN	FN	Precis	TPR	TNR	F1	Accur
funct	26409	6565	20315	5612	0,80090	0,82474	0,75577	0,81265	
non funct	17514	5045	31260	5082	0,77636	0,77509	0,86104	0,77573	
funct nr	1451	1917	52700	2833	0,43082	0,33870	0,96490	0,37925	
Overall									0,77034

Tabla 1: Resultados de C4.5

	Error in %	Size of Test Set	Error Count
fold 0	23,737	11880	2820
fold 1	23,586	11880	2802
fold 2	23,998	11880	2851
fold 3	23,645	11880	2809
fold 4	23,098	11880	2744

Tabla 2: Tabla de error de C4.5

row ID	functional	non functional	functional needs repair
functional	26409	4264	1348
non functional	4513	17514	569
functional needs repair	2052	781	1451

Tabla 3: Matriz de confusión de C4.5

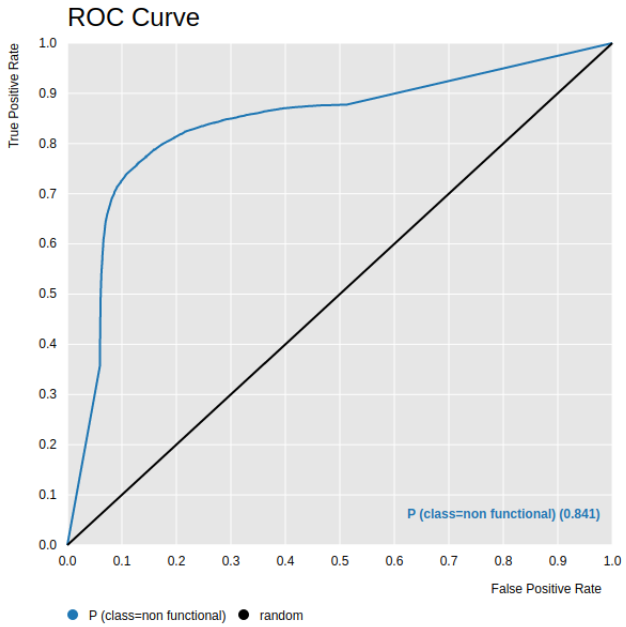

Figura 2: Curva ROC de C4.5

La complejidad de este modelo se mide por el número de hojas. Mediante el nodo "Decision Tree to Ruleset" se puede comprobar que tiene 6368 reglas.

2.2. Naïve-Bayes

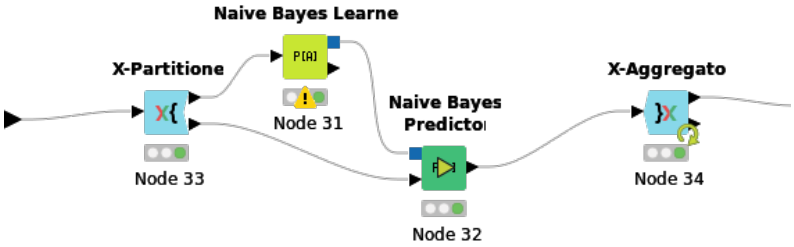

Figura 3: Flujo de trabajo de Naïve-Bayes

	TP	FP	TN	FN	Precis	TPR	TNR	F1	Accur
funct	19275	6325	20816	12984	0,75293	0,59751	0,76696	0,66627	
non funct	14696	7231	29345	8128	0,67022	0,64388	0,80230	0,65679	
funct nr	2159	9714	45369	2158	0,18184	0,50012	0,82365	0,26671	
Overall									0,60825

Tabla 4: Resultados de Naïve-Bayes

	Error in %	Size of Test Set	Error Count
fold 0	39,562	11880	4700
fold 1	41,322	11880	4909
fold 2	36,406	11880	4325
fold 3	40,598	11880	4823
fold 4	37,988	11880	4513

Tabla 5: Tabla de error de Naïve-Bayes

row ID	functional	non functional	functional needs repair
functional	19275	6291	6693
non functional	5107	14696	3021
functional needs repair	1218	940	2159

Tabla 6: Matriz de confusión de Naïve-Bayes

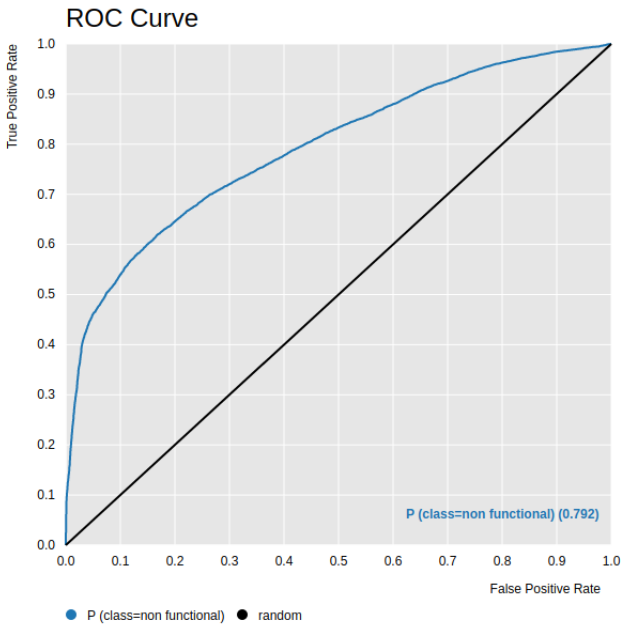

Figura 4: Curva ROC de Naïve-Bayes

2.3. OneR

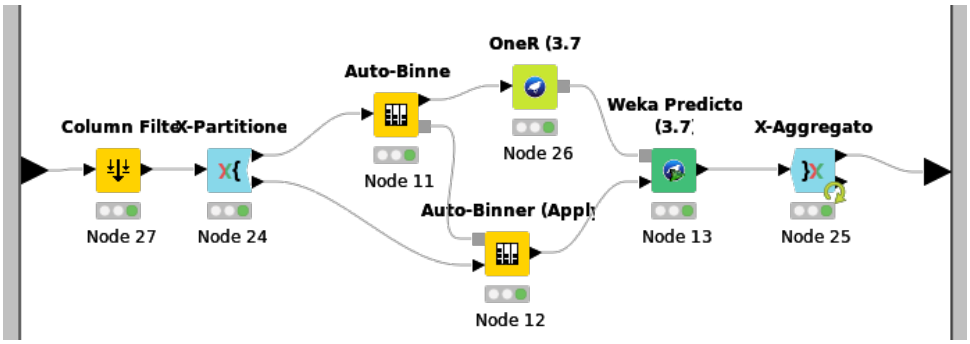

Figura 5: Flujo de trabajo de OneR

	TP	FP	TN	FN	Precis	TPR	TNR	F1	Accur
funct	31889	20476	6665	370	0,60898	0,98853	0,24557	0,75366	
non funct	6614	421	36155	16210	0,94016	0,28978	0,98849	0,44302	
funct nr	0	0	55083	4317		0,00000	1,00000		
Overall									0,64820

Tabla 7: Resultados de OneR

	Error in %	Size of Test Set	Error Count
fold 0	35,067	11880	4166
fold 1	35,177	11880	4173
fold 2	35,37	11880	4202
fold 3	35,455	11880	4212
fold 4	34,832	11880	4138

Tabla 8: Tabla de error de OneR

row ID	functional	non functional	functional needs repair
functional	31889	370	0
non functional	16210	6614	0
functional needs repair	4266	51	0

Tabla 9: Matriz de confusión de OneR

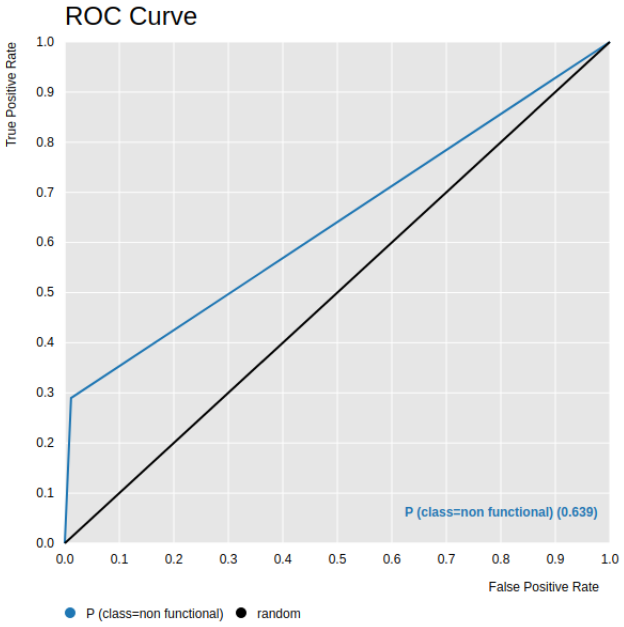

Figura 6: Curva ROC de OneR

Este algoritmo tiene sólo una regla (tal y como indica su propio nombre.

2.4. K-NN

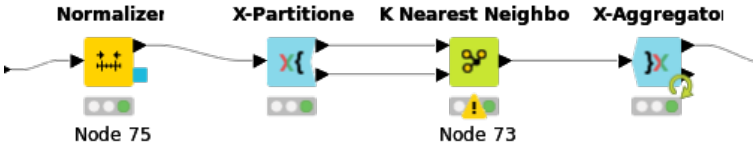

Figura 7: Flujo de trabajo de K-NN

	TP	FP	TN	FN	Precis	TPR	TNR	F1	Accur
funct	24458	9768	17373	7801	0,71460	0,75818	0,64010	0,73574	
non funct	14372	7472	29104	8452	0,65794	0,62969	0,79571	0,64350	
funct nr	1155	2175	52908	3162	0,34685	0,26755	0,96051	0,30208	
Overall									0,67315

Tabla 10: Resultados de K-NN

	Error in %	Size of Test Set	Error Count
fold 0	33,485	11880	3978
fold 1	32,104	11880	3814
fold 2	32,34	11880	3842
fold 3	32,5	11880	3861
fold 4	32,997	11880	3920

Tabla 11: Tabla de error de K-NN

row ID	functional	non functional	functional needs repair
functional	24458	6469	1332
non functional	7609	14372	843
functional needs repair	2159	1003	1155

Tabla 12: Matriz de confusión de K-NN

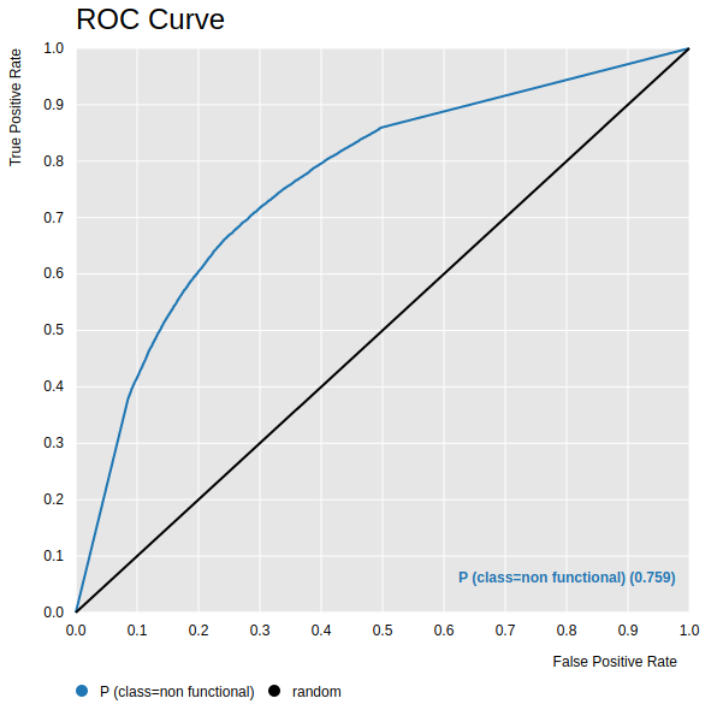

Figura 8: Curva ROC de KNN

En K-NN es imposible determinar la complejidad del algoritmo.

2.5. Random Forest

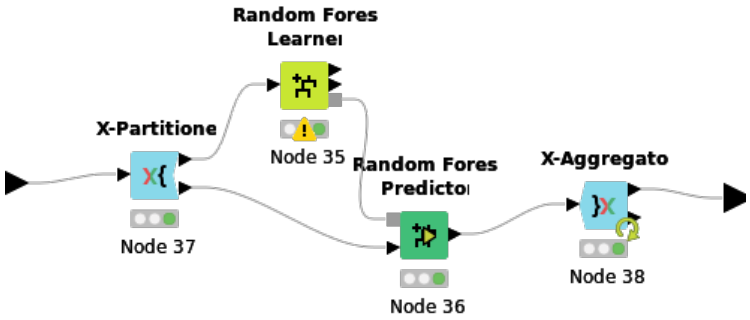

Figura 9: Flujo de trabajo de Random Forest

	TP	FP	TN	FN	Precis	TPR	TNR	F1	Accur
funct	29486	7936	19205	2773	0,78793	0,91404	0,70760	0,84631	
non funct	17125	2891	33685	5699	0,85557	0,75031	0,92096	0,79949	
funct nr	1189	773	54310	3128	0,60601	0,27542	0,98597	0,37872	
Overall									0,80471

Tabla 13: Resultados de Random Forest

	Error in %	Size of Test Set	Error Count
fold 0	19,529	11880	2320
fold 1	19,562	11880	2324
fold 2	19,705	11880	2341
fold 3	19,588	11880	2327
fold 4	19,259	11880	2288

Tabla 14: Tabla de error de Random Forest

row ID	functional	non functional	functional needs repair
functional	29486	2292	481
non functional	5407	17125	292
functional needs repair	2529	599	1189

Tabla 15: Matriz de confusión de Random Forest

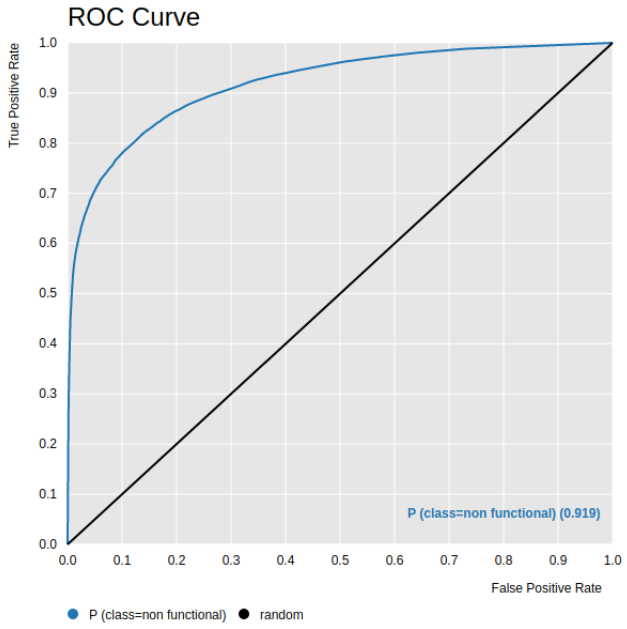

Figura 10: Curva ROC de Random Forest

2.6. Gradient Boosted

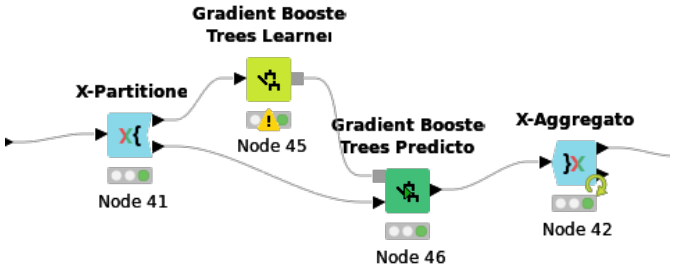

Figura 11: Flujo de trabajo de Gradient Boosted

	TP	FP	TN	FN	Precis	TPR	TNR	F1	Accur
funct	29426	9851	17290	2833	0,74919	0,91218	0,63704	0,82269	
non funct	15666	3274	33302	7158	0,82714	0,68638	0,91049	0,75022	
funct nr	754	429	54654	3563	0,63736	0,17466	0,99221	0,27418	
Overall									0,77182

Tabla 16: Resultados de Gradient Boosted

	Error in %	Size of Test Set	Error Count
fold 0	22.92929292929293	11880	2724
fold 1	22.66835016835017	11880	2693
fold 2	23.114478114478114	11880	2746
fold 3	22.59259259259259	11880	2684
fold 4	22.786195286195287	11880	2707

Tabla 17: Tabla de error de Gradient Boosted

row ID	functional	non functional	functional needs repair
functional	29426	2575	258
non functional	6987	15666	171
functional needs repair	2864	699	754

Tabla 18: Matriz de confusión de Gradient Boosted

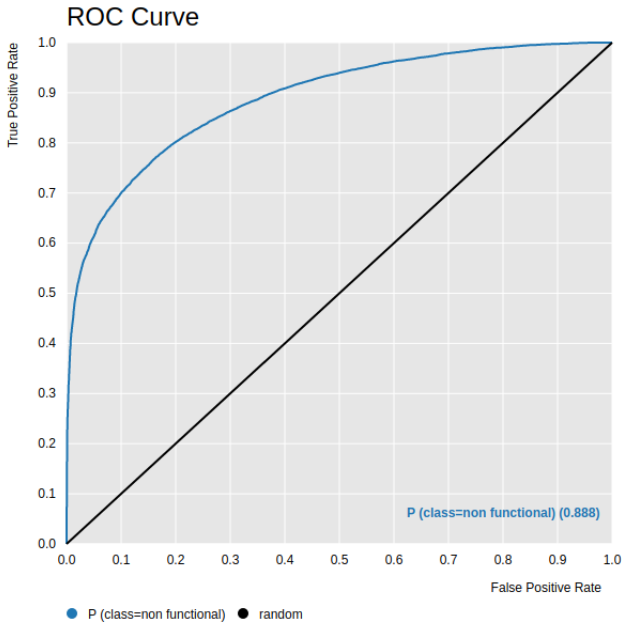

Figura 12: Curva ROC de Gradient Boosted

3. Análisis de resultados

A continuación se muestra una tabla resumen con todos los algoritmos, centrándonos en la clase positiva (non functional). Se han añadido las medidas "Accuracy" (precisión para la clase positiva), "G-mean" (media geométrica de TPR y TNR) y "AUC" (Área Bajo la Curva –Area Under the Curve–). También se muestra una figura con la curva ROC de cada algoritmo.

Algoritmo	**TP**	**FP**	**TN**	**FN**	**TPR**	**TNR**	**PPV**
C4.5	17514	5045	31260	5082	0,7751	0,8610	0,7764
Naïve-Bayes	14696	7231	29345	8128	0,6439	0,8023	0,6702
OneR	6614	421	36155	16210	0,2898	0,9885	0,9402
K-NN	14372	7472	29104	8452	0,6297	0,7957	0,6579
Random Forest	17125	2891	33685	5699	0,7503	0,9210	0,8556
Gradient Boosted	15666	3274	33302	7158	0,6864	0,9105	0,8271

Tabla 19: Resumen (1) de resultados de todos los algoritmos

Algoritmo	Accuracy	F1-score	G-mean	AUC
C4.5	0,8281	0,7757	0,8169	0,8405
Naïve-Bayes	0,7414	0,6568	0,7187	0,7925
OneR	0,7200	0,4430	0,5352	0,6391
K-NN	0,7319	0,6435	0,7078	0,7590
Random Forest	0,8554	0,7995	0,8313	0,9190
Gradient Boosted	0,8244	0,7502	0,7905	0,8885

Tabla 20: Resumen (2) de resultados de todos los algoritmos

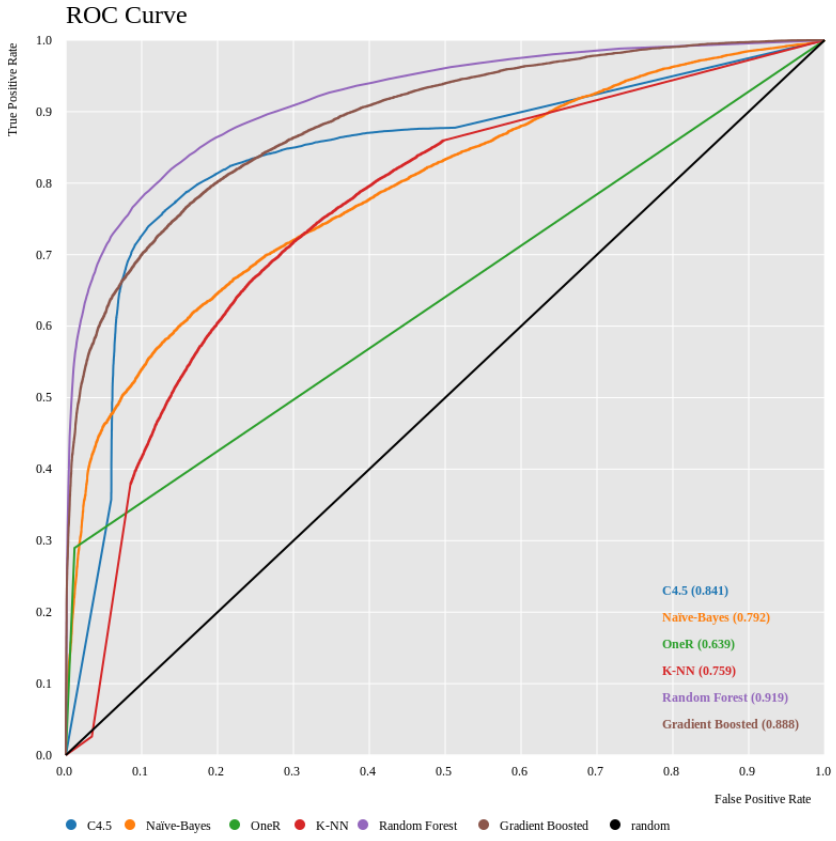

Figura 13: Curvas ROC de todos los algoritmos

A priori se pueden diferenciar dos grupos: uno con resultados relativamente malos, conformado por OneR, K-NN y Naïve-Bayes, y otro que ofrece mejor rendimiento, formado por el resto de algoritmos.

Era de esperar que OneR tuviera los valores más bajos de la segunda tabla, ya que con una única regla poca precisión se puede alcanzar. Siguiendo con esta medida, Naïve-Bayes no alcanza una gran precisión debido a que quizás las variables del dataset estén relacionadas, y este algoritmo penaliza la correlación entre ellas. K-NN es muy probable que necesite aumentar la distancia con el número vecinos para poder obtener un modelo mejor.

En cuanto al grupo de los tres mejores hay poca diferencia en los resultados, aunque destaca levemente Random Forest ya que es un algoritmo que trata bien los valores perdidos y halla correlaciones entre variables encontrando las más relevantes, de modo que tiene sin optimización alguna una tasa de aciertos muy superior al resto. Gradient Boosted, dada su robustez y flexibilidad, consigue una buena precisión y AUC, y sorprende que C4.5 logre resultados similares, seguramente porque lidia muy bien contra el ruido y los valores perdidos.

Por último cabe destacar que los valores de TPR son inferiores a los de TNR, lo cual es un indicativo de que existe un gran desbalanceo de las clases. Esto nos hace ver que C4.5 y Random Forest sufren menos ante tal desbalanceo, algo lógico teniendo en cuenta que son algoritmos que en la literatura se muestran robustos ante este problema. Sin embargo, Gradient Boosted, que ofrece buenos resultados, se muestra débil ante el desbalanceo, probablemente porque no realiza una buena generalización tras el boosting.

4. Configuración de algoritmos

4.1. C4.5

Con los nodos de "Parameter Optimization" he variado "min number records per node" en el nodo "Decision Tree Learner" con valores entre 2 y 15, combinando los dos valores de "quality measure" y "pruning method" para maximizar la precisión. El resultado ha sido que se obtiene mejor precisión con "gini index" y "no pruning" con un valor de 8 en "min number records per node".

Sin embargo, tal y como se puede ver en la tabla, la diferencia de los mejores resultados de cada configuración indica que podría usarse poda para mejorar la interpretabilidad del árbol sin sacrificar excesivamente la precisión.

		Min number of records	Accuracy
Default	**Gini index No pruning**	2	0,7703
Gini index	**No pruning**	8	0,7757
	MDL	2	0,7716
Gain ratio	**No pruning**	12	0,7695
	MDL	11	0,7676

Tabla 21: Mejores resultados para cada configuración de C4.5

4.2. Naïve-Bayes

Con los nodos de "Parameter Optimization" he variado los valores de "default probability" entre 0,00001 y 1,00001 y de "maximum number of unique nominal values per attribute" con valores entre 13 y 21 en el nodo "Naïve Bayes Learner" para maximizar la precisión. Dado el gran número de combinaciones realizadas sólo se muestran los resultados para "default probability" con los valores 0,00001, 0,05 y 1, puesto que todos los resultados son muy similares.

Max Nom Values	**Def. Prob.**	**Accuracy**	**Def. Prob.**	**Acc.**	**Def. Prob.**	**Acc.**
Default (20)	1E-04	0,608				
13	1E-05	0,6279	0,05	0,67916	1	0,67449
14	1E-05	0,6279	0,05	0,67916	1	0,67449
15	1E-05	0,6279	0,05	0,67916	1	0,67449
16	1E-05	0,6279	0,05	0,67916	1	0,67449
17	1E-05	0,6279	0,05	0,67916	1	0,67449
18	1E-05	0,6318	0,05	0,67369	1	0,66655
19	1E-05	0,6318	0,05	0,67369	1	0,66655
20	1E-05	0,6318	0,05	0,67369	1	0,66655
21	1E-05	0,6242	0,05	0,66554	1	0,66135

Tabla 22: Resultados de Naïve-Bayes

Un valor de "default probability" bajo, pero no cercano a cero, mejora la precisión. A su vez, el número máximo de valores nominales da resultados iguales en la mayoría de ocasiones en el rango de 13 a 17, donde arroja mayor precisión. Los mejores valores paramétricos se han obtenido con 13 valores nominales como máximo y una probabilidad por defecto de 0,05.

4.3. OneR

No presenta ningún tipo de mejora, ya que utiliza siempre la misma regla.

4.4. K-NN

Como en el resto de algoritmos, he utilizado "Parameter Optimization" para encontrar el valor óptimo de k-vecinos, variando entre 1 y 15, para maximizar la precisión. El número de vecinos óptimo es 14. En este caso la complejidad del modelo no es relevante, ya que al aumentar el número de vecinos sólo aumenta el cómputo.

Número de vecinos	**Accuracy**
Default (3)	0,673
1	0,75372
2	0,75423
3	0,76892
4	0,77268
5	0,77655
6	0,77855
7	0,77976
8	0,78
9	0,78103
10	0,78163
11	0,78163
12	0,78177
13	0,7817
14	0,78189
15	0,78187

Tabla 23: Mejores resultados para cada vecino de K-NN

4.5. Random Forest

Para encontrar los parámetros óptimos del algoritmo he comenzado iterando "split criterion" con la opción "Information Gain Ratio" con diferentes

números de modelos y límites en cuanto al número de niveles. Los valores por defecto de este algoritmo son "Gini Index" con 100 modelos y sin límite en el número de niveles, dando una precisión de 0,805.

Núm. modelos	**Límit. núm. niveles**	**Accuracy**
5	5	0,698
50	5	0,70345
100	5	0,70635
150	5	0,70699
200	5	0,70673
250	5	0,70687
50	10	0,71852
100	10	0,71761
150	10	0,7183
200	10	0,71867
250	10	0,71848
50	15	0,73379
100	15	0,73362
150	15	0,73375
200	15	0,73407
250	15	0,73404
200	—	0,80633

Tabla 24: Resultados de Random Forest con Information Gain Ratio

Las diferencias en los resultados son mínimas y se puede comprobar que el número de modelos a utilizar no afecta tanto al resultado como sí lo hace el número de niveles máximo, siendo mejor el resultado cuanto mayor es el límite. Utilizando 200 modelos sin límite de niveles el incremento de precisión es bastante más notorio.

Con "Gini index" he realizado las mismas iteraciones, pero en vista de los resultados anteriores he decidido no comprobar el número de modelos ni el límite al número de niveles con valores de 50 ni 5, respectivamente.

Núm. modelos	**Límit. núm. niveles**	**Accuracy**
5	5	0,719
100	10	0,76608
150	10	0,76487
200	10	0,76582
250	10	0,76559
100	15	0,79648
150	15	0,79613
200	15	0,79633
250	15	0,79658

Tabla 25: Resultados de Random Forest con Gini Index

Se puede comprobar que para los mismos valores en cada parámetro "Gini index" arroja mejores resultados que "Information Gain Ratio", pero al igual que antes no es el número de modelos el que más afecta al resultado, sino el límite del número de niveles. Aún así no se ha conseguido ni si quiera alcanzar la precisión obtenida con el valor por defecto del algoritmo (sólo al no utilizar límite en el número de niveles con "Gain Ratio"), y por ello, siguiendo con "Gini index" he eliminado el límite al número de niveles y comprobado cómo variaba la precisión conforme aumentaba el número de modelos desde 100 hasta 110 con incremento de 5; hasta 150 con incrementos de 10; y hasta 300 con saltos de 50.

Núm. modelos	**Accuracy**
100	0,80956
105	0,80911
110	0,80919
120	0,80916
130	0,80889
140	0,80894
150	0,80911
200	0,80929
250	0,80973
300	0,8098

Tabla 26: Resultados de Random Forest con Gini Index sin límite en el número de niveles

La diferencia de precisión entre el valor más alto y el más bajo es del 0,1 %. No hay un aumento lineal respecto al incremento en el número de

modelos, excepto a partir de los 200, donde sí se logran valores más altos, obteniéndose la mejor precisión con 300 modelos, aunque la diferencia con el valor obtenido con 100 modelos es del 0,024 %.

Gini Index	**Núm. modelos**	**Accuracy**
Por defecto	100	0,805
Optimización	300	0,8098

Tabla 27: Resultados de Random Forest

En resumen, aumentando el número de modelos se logra mejorar algo el resultado, pero es el límite en el número de niveles donde más se gana, aunque conlleva perder intepretabilidad, por lo que con pocos modelos y limitando el número de niveles a 10 se podría conseguir un buen resultado con cierta interpretabilidad del modelo.

4.6. Gradient Boosted Trees

Los parámetros a configurar con este algoritmo han sido el número de modelos, el límite al número de niveles (profundidad del árbol) y la tasa de aprendizaje. Por defecto el algoritmo establece estos valores en 100 modelos, 4 niveles y 0,1 la tasa de aprendizaje. Para la optimización he variado el número de modelos entre 100 y 300 en saltos de 50; el número de niveles máximo lo he establecido en 4, 7 y 10; por último, la tasa de aprendizaje la he establecido en 0,1 y 0,05.

Num. modelos	**Num. niveles**	**Learn. rate**	**Accur.**	**Num. modelos**	**Num. niveles**	**Learn. rate**	**Accur.**
Default (100)	4	0,1	0,772				
100	4	0,1	0,78253	100	4	0,05	0,77227
150	4	0,1	0,78717	150	4	0,05	0,77806
200	4	0,1	0,78998	200	4	0,05	0,78120
250	4	0,1	0,79318	250	4	0,05	0,78396
300	4	0,1	0,79497	300	4	0,05	0,78648
100	7	0,1	0,80145	100	7	0,05	0,79374
150	7	0,1	0,80402	150	7	0,05	0,79911
200	7	0,1	0,80537	200	7	0,05	0,80157
250	7	0,1	0,80551	250	7	0,05	0,80318
300	7	0,1	0,80530	300	7	0,05	0,80391
100	10	0,1	0,80808	100	10	0,05	0,80621
150	10	0,1	0,80790	150	10	0,05	0,80796
200	10	0,1	0,80822	200	10	0,05	0,80843
250	10	0,1	0,80746	250	10	0,05	0,80803
300	10	0,1	0,80621	300	10	0,05	0,80786
600	10	0,1	0,80429	600	10	0,05	0,80675
800	10	0,1	0,80365	800	10	0,05	0,80692

Tabla 28: Resultados de Gradient Boosted Trees

Tras ejecutar todas estas combinaciones se puede comprobar que aumentar el número de modelos no aumenta significativamente la precisión; de hecho, tras comprobar que a partir de los 200 modelos la precisión descendía ligeramente, probé con 600 y 800 modelos y ésta seguía descendiendo. El aumento de la profundidad del árbol sí que influye algo más en dicho aumento, aunque hace que el modelo resultante sea más complejo. La tasa de aprendizaje, tal y como indica la propia descripción del algoritmo en la documentación de KNIME, debería mejorar los resultados cuanto menor sea este valor al aumentar el número de modelos. En efecto, los resultados son algo mejores con una tasa de 0,05 en lugar de 0,1 para un número de modelos mayor, pero sólo en aquellos casos donde el número de niveles máximo también es más grande. Una vez analizadas todas estas características he escogido para la optimización 200 modelos, 10 niveles de profundidad como máximo y una tasa de aprendizaje de 0,05.

5. Procesados de datos

En primer lugar he podido comprobar mediante histogramas que dentro de cada variable los valores no suelen encontrarse repartidos de manera uniforme dentro del conjunto. Algunos ejemplos son:

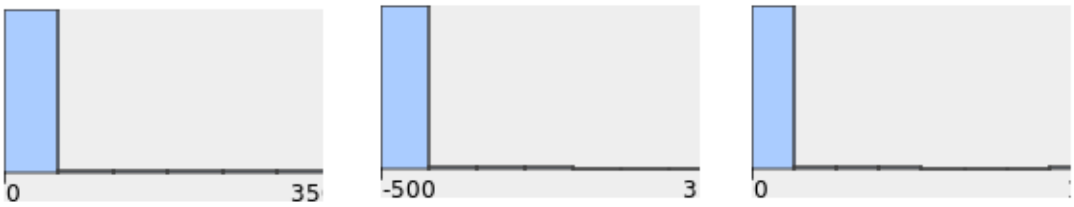

Figura 14: Histogramas de amount_tsh, district_code y num_private

Mediante una tabla con las correlaciones lineales se pueden observar cuáles son las variables que más influyen en el resultado de la clase. Aquellas que presenten una correlación nula podrán ser buenas candidatas a ser eliminadas si así fuera necesario.

Habiendo hecho un análisis del conjunto de datos he podido determinar que hay muchas variables que parece razonable eliminar por diversos motivos, muchos de los cuales son comunes en varios de dichos atributos. Apoyándome además en su importancia en cuanto a la matriz de correlaciones anterior se describen a continuación cuáles he decidido quitar y los motivos:

- date_recorded: parece irrelevante para el resultado en qué fecha se tomó el dato, haciendo que la cantidad de valores distintos sea de 356.
- funder: presenta más de 1000 valores únicos y 3662 valores perdidos (6,16 %).
- installer: contiene más de 1000 valores distintos y 3696 valores perdidos (6,22 %).
- wpt_name: presenta más de 1000 valores únicos y 3575 valores perdidos (6,02 %).
- num_private: no tiene ninguna descripción en www.drivendata.org y tiene un valor igual a 0 en el 98,73 % de las filas.
- subvillage: tiene más de 17000 valores distintos.

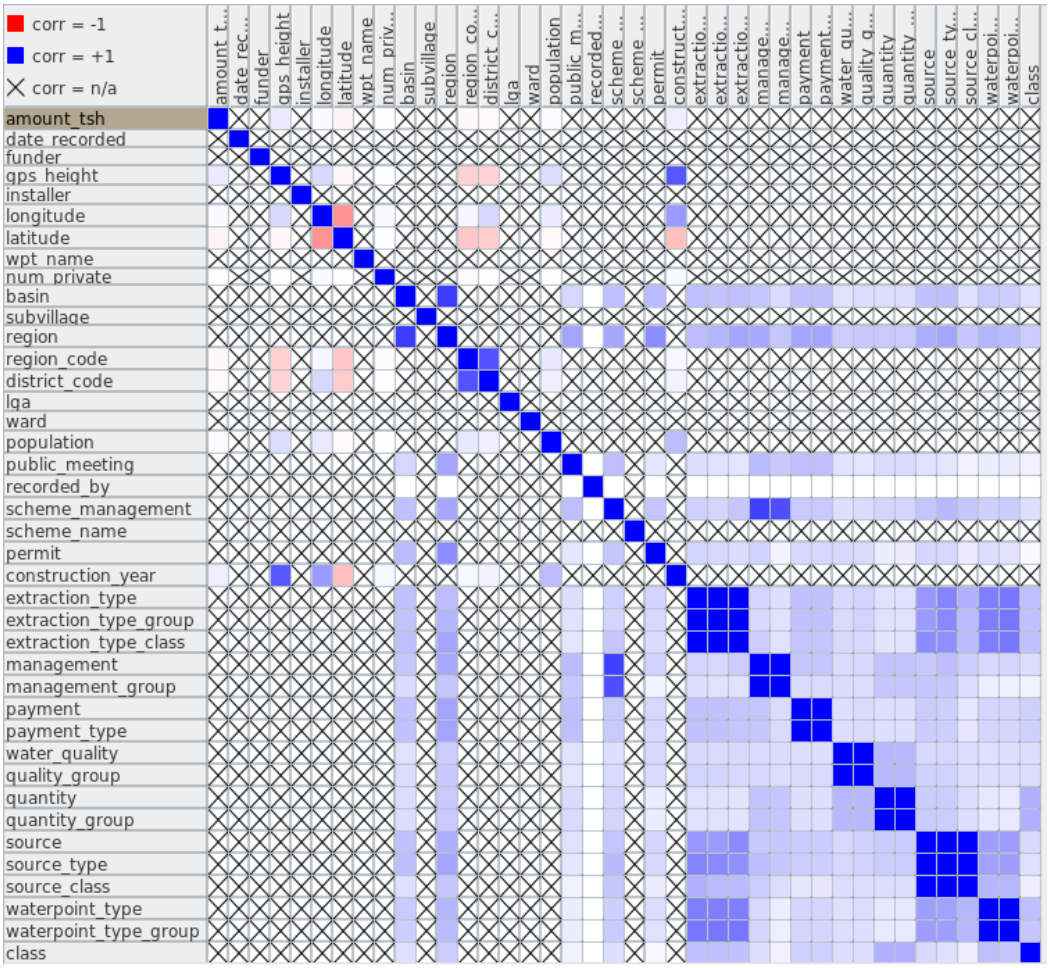

Figura 15: Matriz de correlaciones

- region_code: se corresponde con region y además tiene fallos de codificación (los valores 90 y 99 se corresponden con 9; 80 con 8; 60 con 6; y 24 con 2.
- ward: presenta más de 1000 valores únicos. Además cada ward se incluye en un lga.
- recorded_by: sólo contiene un único valor.
- scheme_name: presenta más de 1000 valores únicos y 28174 valores perdidos (47,43 %).
- payment: es igual que payment_type.
- quantity_group: es igual que quantity.

Para eliminar dichas columnas he hecho uso de un “Column Filter” entre el “File Reader” y la entrada al metanodo de cada algoritmo.

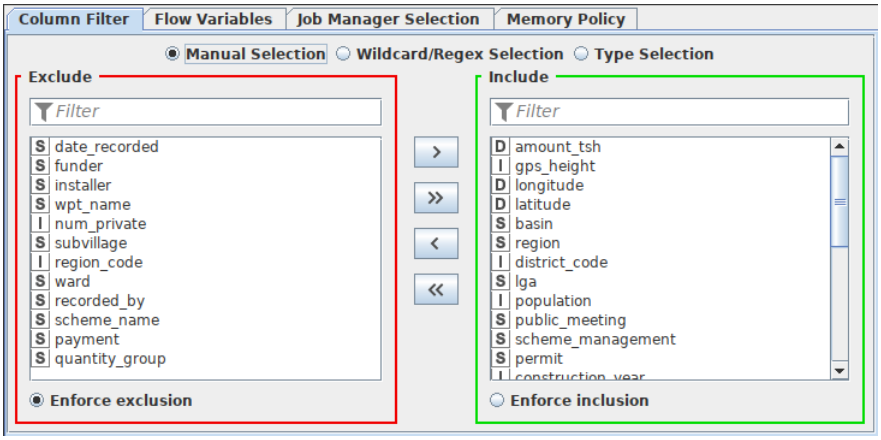

Figura 16: Filtrado de columnas

Hay otras variables que también son susceptibles de no ser consideradas, ya que tienen aproximadamente un 35 % de valores igual a cero, como son gps_height, population y construction_year, y con un 70 % amount_tsh. No está muy claro el motivo de tener un cero en cada celda, por lo que he decidido mantener estas columnas.

5.1. C4.5

Se ha realizado una imputación básica de valores perdidos: la media para los números enteros y decimales, y el valor más frecuente para las cadenas de texto.

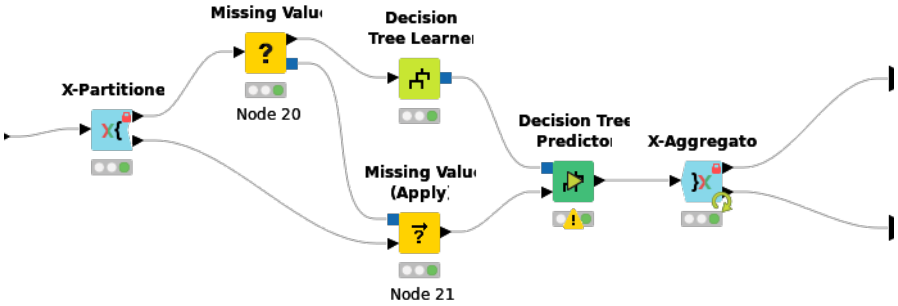

Figura 17: Preprocesado de C4.5

Los resultados de aplicar dicha optimización son:

	TP	**FP**	**TN**	**FN**	**TPR**	**TNR**
C4.5	17514	5045	31260	5082	0,7751	0,8610
C4.5 (opt) + [prep]	17178	4545	31872	5500	0,7575	0,8752

Tabla 29: Comparación (1) de C4.5 por defecto vs. optimizado+preprocesado

	PPV	**Accuracy**	**F1-score**	**G-mean**	**AUC**
C4.5	0,7764	0,8281	0,7757	0,8169	0,8405
C4.5 (opt) + [prep]	0,7908	0,8300	0,7738	0,8142	0,8830

Tabla 30: Comparación (2) de C4.5 por defecto vs. optimizado+preprocesado

Estos resultados, aunque mínimos, son ligeramente mejores que los resultados por defecto, aunque se aprecia una pequeña disminución del TPR, F1-score y G-mean, debido a que la cantidad de "True Positives" es también menor. A pesar de esto aumenta en unas décimas el área bajo la curva.

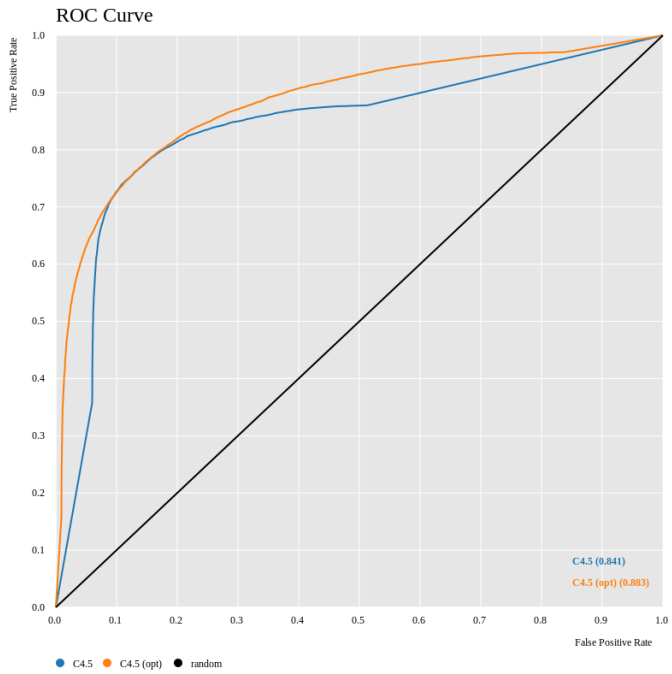

Figura 18: Comparación Curva ROC C4.5

5.2. Naïve-Bayes

He añadido un nodo "Column Filter" para eliminar la variable "lga", puesto que tiene demasiados valores distintos para este algoritmo. Además he realizado una imputación de valores perdidos igual que la realizada con C4.5.

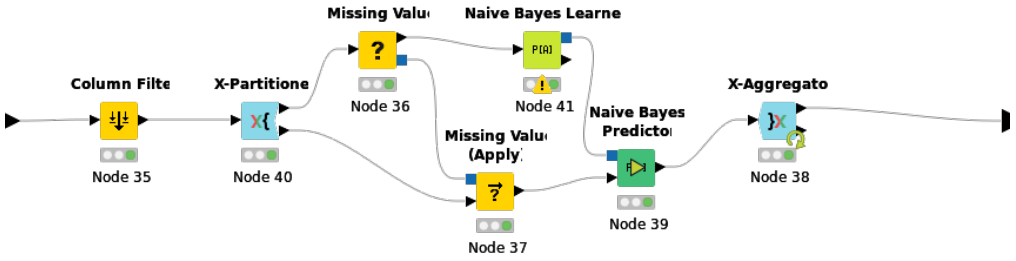

Figura 19: Preprocesado de Naïve-Bayes

	TP	**FP**	**TN**	**FN**	**TPR**	**TNR**
C4.5	17514	5045	31260	5082	0,7751	0,8610
C4.5 (opt) + [prep]	17178	4545	31872	5500	0,7575	0,8752

Tabla 31: Comparación (1) de Naïve-Bayes por defecto vs. optimizado+preprocesado

	PPV	**Accuracy**	**F1-score**	**G-mean**	**AUC**
Naïve-Bayes	0,6702	0,7414	0,6568	0,7187	0,7925
Naïve-Bayes (opt) + [prep]	0,7430	0,7608	0,6498	0,7109	0,7934

Tabla 32: Comparación (2) de Naïve-Bayes por defecto vs. optimizado+preprocesado

Al igual que ha ocurrido con C4.5, hay un descenso en la tasa de TPR, F1-socre y prácticamente el mismo resultado en G-mean y AUC; sí que mejora PPV y la precisión.

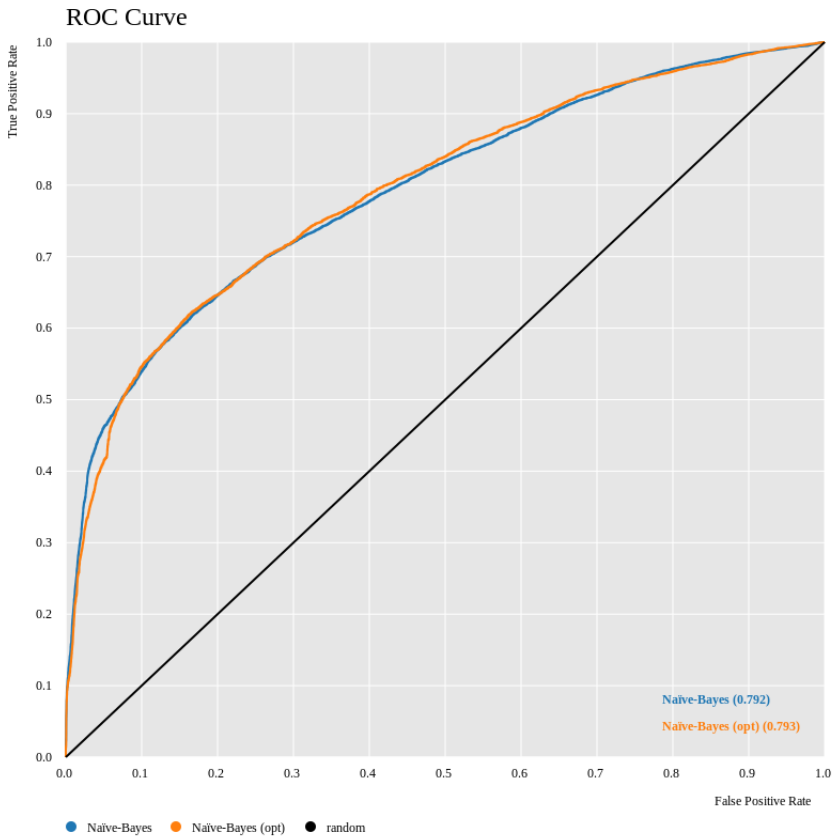

Figura 20: Comparación Curva ROC Naïve-Bayes

5.3. OneR

He utilizado un nodo "Column Filter" para eliminar, por la misma razón que con el algoritmo de Naïve-Bayes, la variable "lga".

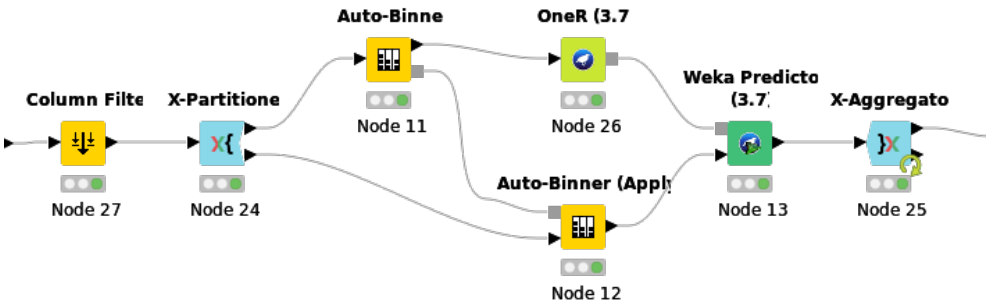

Figura 21: Preprocesado de OneR

Como la única regla utilizada es “quantity” el modelo queda exactamente igual.

5.4. K-NN

En primer lugar utilizo “Category to Number” para tratar con “lga”, estableciendo el máximo de categorías a 125. El siguiente paso es utilizar “One to Many” para tratar con el resto de variables categóricas debido a que presentan un número menor de variables distintas y no hacen crecer extremadamente el dataset.

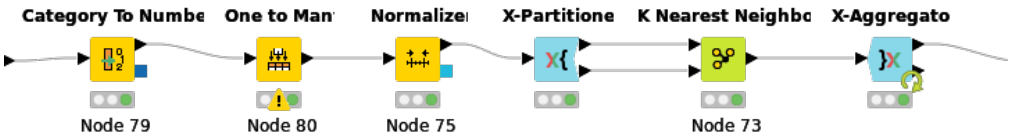

Figura 22: Preprocesado de K-NN

	TP	**FP**	**TN**	**FN**	**TPR**	**TNR**
KNN	14372	7472	29104	8452	0,6297	0,7957
KNN (opt) + [prep]	17228	4169	32407	5596	0,7548	0,8860

Tabla 33: Comparación (1) de K-NN por defecto vs. optimizado+preprocesado

	PPV	**Accuracy**	**F1-score**	**G-mean**	**AUC**
KNN	0,6579	0,7319	0,6435	0,7078	0,7590
KNN (opt) + [prep]	0,8052	0,8356	0,7792	0,8178	0,8906

Tabla 34: Comparación (2) de K-NN por defecto vs. optimizado+preprocesado

Mejoran todas las medidas analizadas.

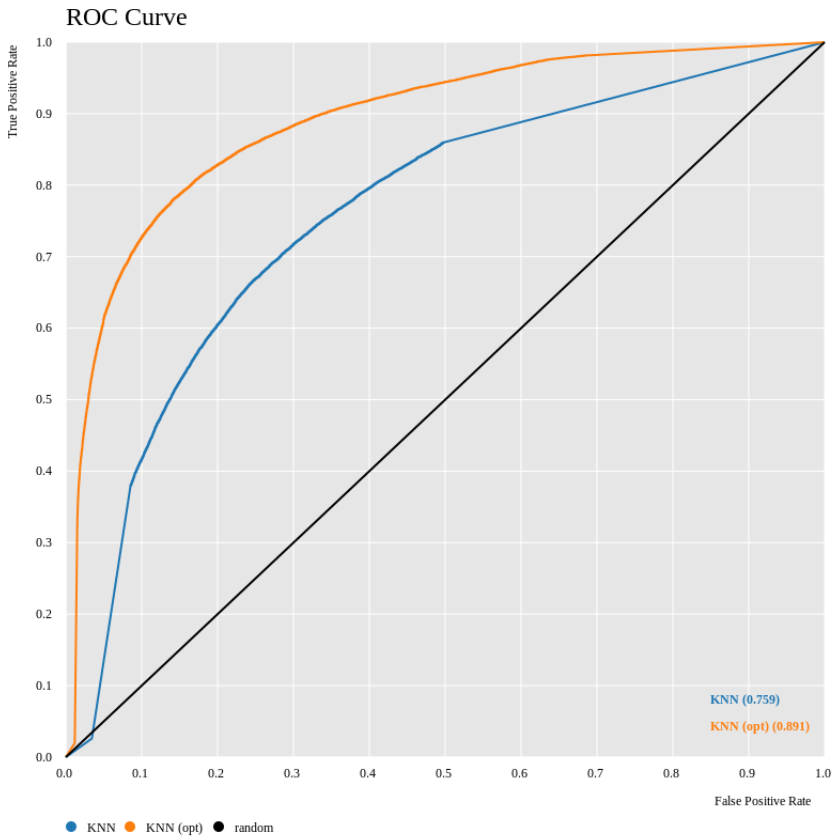

Figura 23: Comparación Curva ROC K-NN

5.5. Random Forest

Para evitar un "warning" he utilizado un nodo "Domain Calculator" para averiguar todos los posibles valores de todas las variables categóricas con menos de 125 valores distintos, y el mínimo y máximo en variables numéricas.

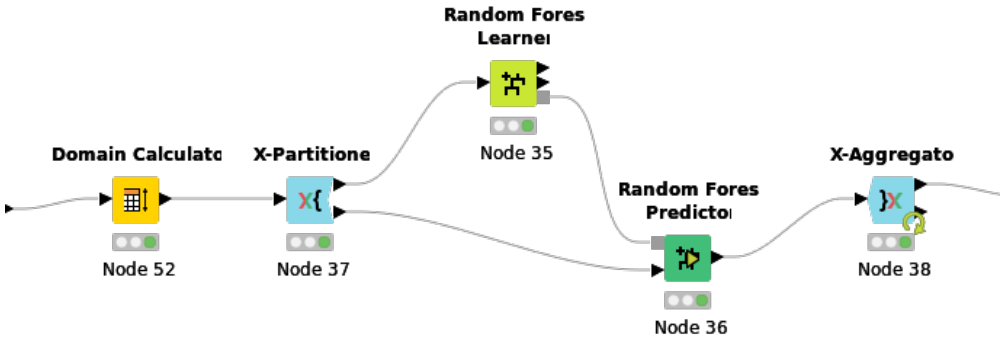

Figura 24: Preprocesado de Random Forest

	TP	FP	TN	FN	TPR	TNR
Random Forest	17125	2891	33685	5699	0,7503	0,9210
Random Forest (opt) + [prep]	17398	2863	33713	5426	0,7623	0,9217

Tabla 35: Comparación (1) de Random Forest por defecto vs. optimizado+preprocesado

	PPV	Accuracy	F1-score	G-mean	AUC
Random Forest	0,8556	0,8554	0,7995	0,8313	0,9190
Random Forest (opt) + [prep]	0,8587	0,8605	0,8076	0,8382	0,9231

Tabla 36: Comparación (2) de Random Forest por defecto vs. optimizado+preprocesado

Se consigue una leve mejora de todas las medidas.

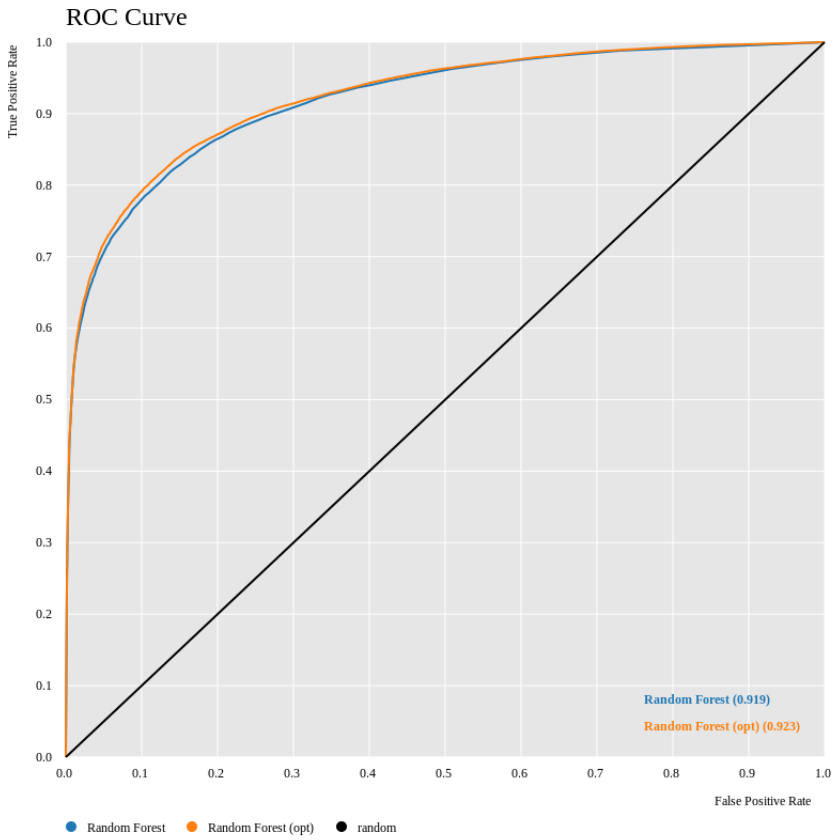

Figura 25: Comparación Curva ROC Random Forest

5.6. Gradient Boosted

He seguido el mismo preproceso que con Random Forest.

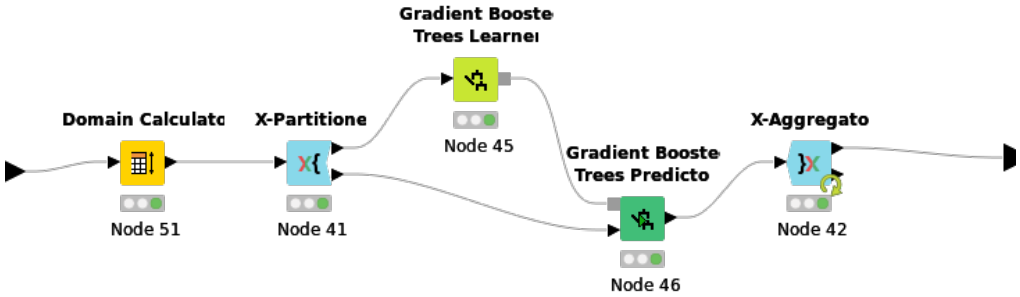

Figura 26: Preprocesado de Gradient Boosted

	TP	FP	TN	FN	TPR	TNR
Gradient Boosted	15666	3274	33302	7158	0,6864	0,9105
Gradient Boosted (opt) + [prep]	17756	3332	33244	5068	0,7780	0,9089

Tabla 37: Comparación (1) de Gradient Boosted por defecto vs. optimizado+preprocesado

	PPV	Accuracy	F1-score	G-mean	AUC
Gradient Boosted	0,8271	0,8244	0,7502	0,7905	0,8885
Gradient Boosted (opt) + [prep]	0,8420	0,8586	0,8087	0,8409	0,9239

Tabla 38: Comparación (2) de Gradient Boosted por defecto vs. optimizado+preprocesado

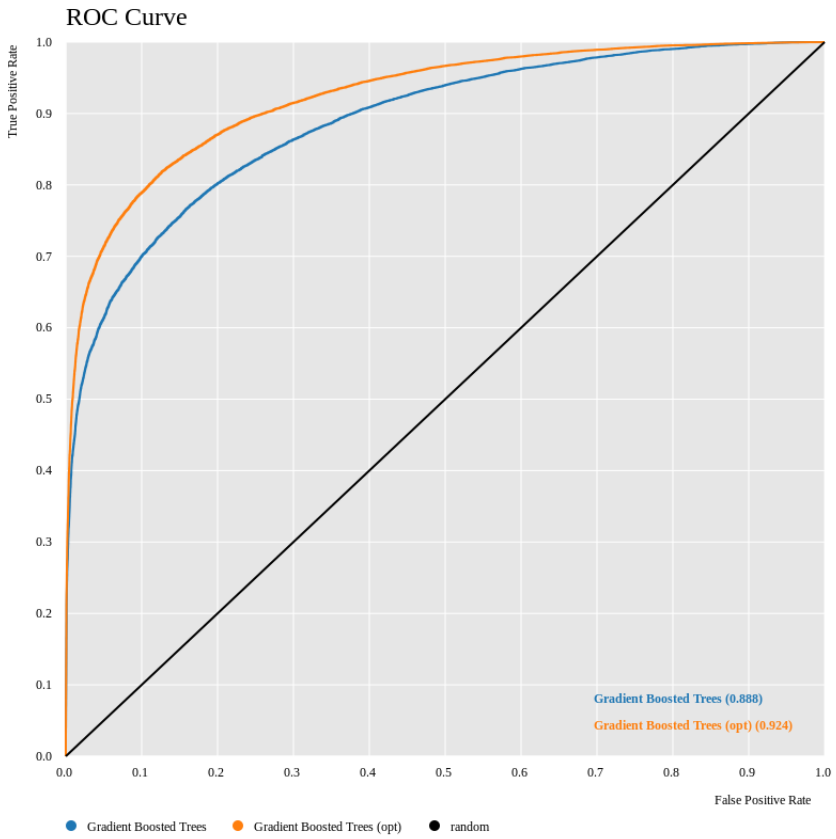

Figura 27: Comparación Curva ROC Gradient Boosted

Mejoran todas la variables analizadas, exceptuando TN y TNR.

6. Interpretación de resultados

6.1. Análisis general de resultados

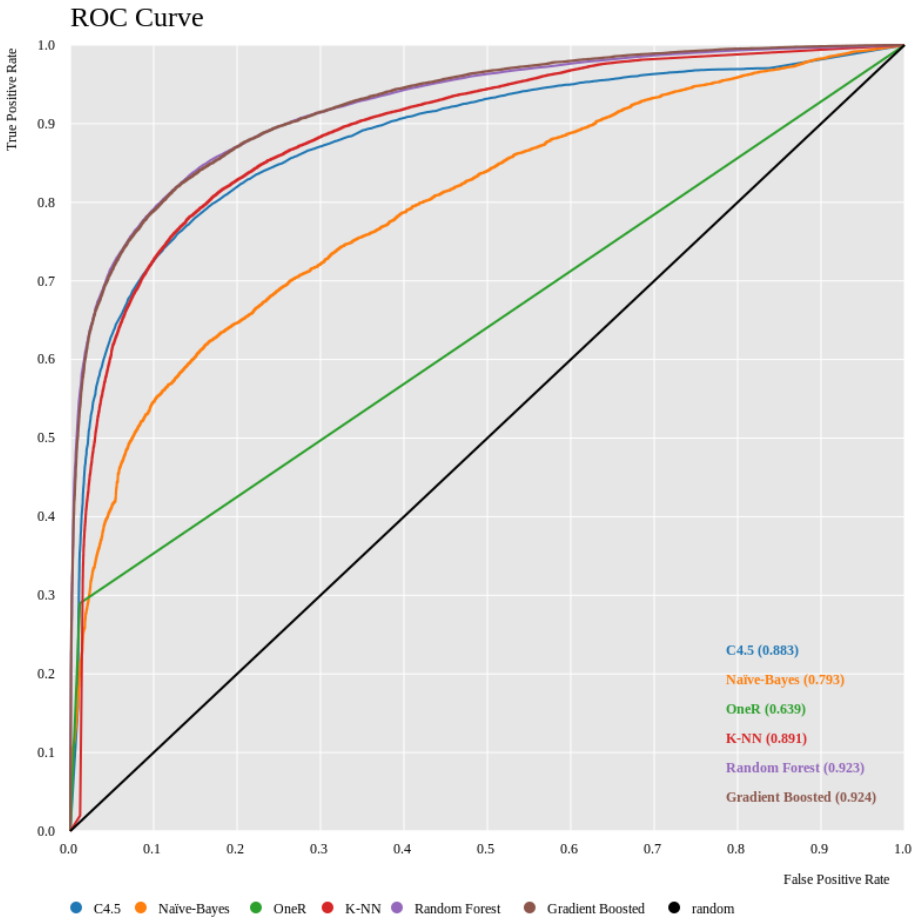

Figura 28: Comparación Curva ROC algoritmos optimizados con preprocesado

Los resultados parecen indicar que para los algoritmos C4.5 y Naïve-Bayes la optimización y el preprocesado mejoran por lo general sus resultados, pero la eliminación de ciertas variables hacen descender la tasa de aciertos, aunque globalmente mejoran el modelo.

En K-NN la mejora es muy buena debido a que se ha conseguido encontrar un número de vecinos óptimo para poder predecir a qué clase pertenece cada instancia, logrando ser así el tercer algoritmo con mejores resultados entre los analizados y del cual se ha logrado una mejora mayor.

En Random Forest apenas se consigue mejora debido a que es un algoritmo muy potente por sí solo que no necesita de mucho ajuste paramétrico para alcanzar obtener el máximo rendimiento.

Por último Gradient Boosted consigue una ligera mejora que hace que obtenga los mejores resultados en comparación con el resto de algoritmos, aunque queda prácticamente empatado con Random Forest a costa de haber hecho más complejo el modelo.

6.2. Modelos generados

El algoritmo C4.5 elige como primer nodo la variable "quantity".

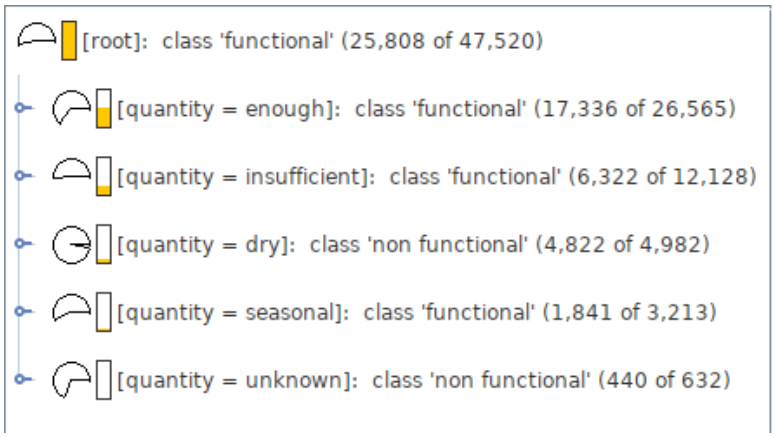

Figura 29: Primer nodo del árbol de C4.5

La única regla utilizada por OneR también es la variable "quantity".

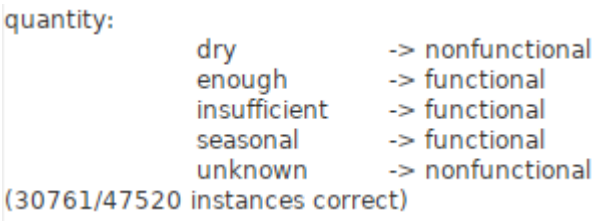

Figura 30: Única regla de OneR

A continuación se muestran los "splits" más frecuentes en Random Forest

Row ID	#splits (lvl 0)	#splits (lvl 1)	#splits (lvl 2)
quantity	41	99	149
region	12	42	100
waterpoint_type	44	43	84
lga	13	44	65
payment_type	8	45	64
extraction_type	36	34	60
construction_year	14	20	59
waterpoint _type_group	51	47	55
source	2	37	52
amount_tsh	14	24	52
source_type	0	13	52
extraction_ type_group	31	44	51
extraction_ type_class	24	24	47
management	4	20	42
basin	1	5	41
scheme_management	1	7	35
longituide	0	6	32
quality_group	1	11	29
latitude	0	6	24

Tabla 39: Número de "splits" más frecuentes en Random Forest

Se puede apreciar por tanto que la variable "quantity" es la más importante para determinar en primera instancia cómo clasificar la clase a la que pertenece cada fila.

6.3. WorkFlow

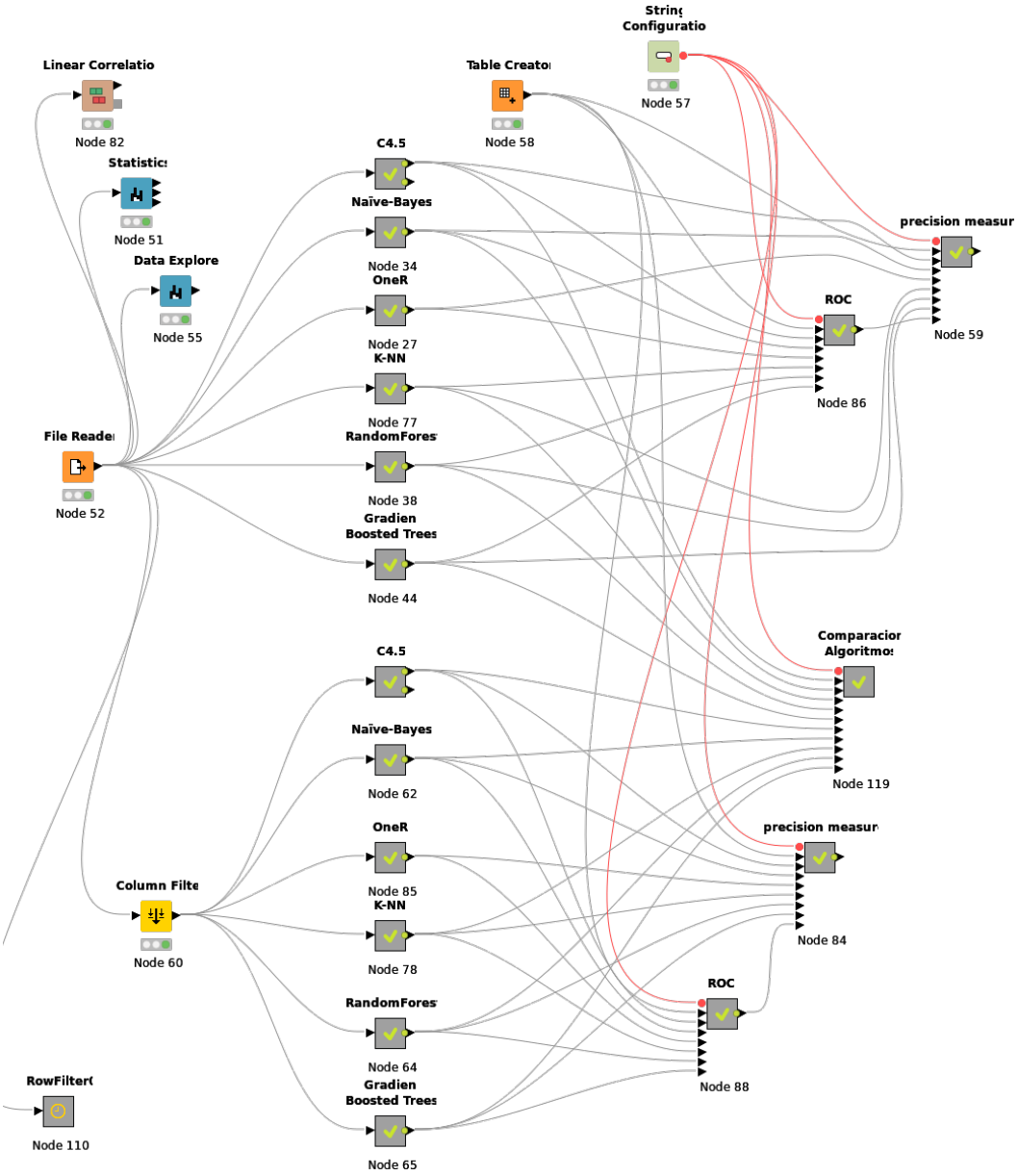

Figura 31: Flujo de trabajo general

7. Contenido adicional

8. Bibliografía

1. KNIME
2. NodePit
3. Algoritmos de aprendizaje: KNN & KMEANS
4. Ensambladores: Random Forest
5. Gradient Boosting Machines
6. Pros and cons of random forests
7. Decision Trees – C4.5
8. Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset