

Inteligencia de Negocio (2019-2020)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Memoria Práctica 1

Análisis Predictivo Mediante Clasificación

Daniel Terol Guerrero
DNI: 09076204J
Correo: danielterol@correo.ugr.es

4 de noviembre de 2019

ÍNDICE

1. Introducción	1
2. Resultados obtenidos	2
2.1. C4.5	2
2.2. Random Forest	3
2.3. Gradient Boosted	5
2.4. Fuzzy Rules	6
2.5. SVM	7
2.6. Naïve Bayes	9
2.7. RProp MLP	10
3. Análisis de resultados	12
4. Configuración de algoritmos	14
4.1. C4.5	14
4.1.1. C4.5 Gain Ratio	14
4.1.2. C4.5 con poda	14
4.2. Random Forest	16
4.2.1. Random Forest Gini Index	16
4.2.2. Random Forest 200 modelos	17
4.3. RProp MLP	19
4.3.1. MLP con 200 modelos, 3 capas ocultas y 100 neuronas por capa.	19
4.3.2. MLP con 200 modelos, 6 capas ocultas y 50 neuronas por capa.	20
5. Procesado de datos	22
5.1. C4.5	23
5.2. Random Forest	24
5.3. RProp MLP	24
5.4. Naïve Bayes	25
6. Interpretación de resultados	25
7. Bibliografía	26

1. INTRODUCCIÓN

En esta práctica se va a abordar un problema de clasificación que consiste en analizar un *dataset* para poder detectar si hay bombas de agua no funcionales. Para realizar el análisis, se va a abordar un conjunto de datos con 59000 instancias con un conjunto de atributos que determinan si una bomba de agua funciona, no funciona o funciona pero necesita reparación. Por tanto, utilizando diferentes modelos, se va a intentar predecir con cierta probabilidad qué bombas de agua se van a romper antes de ser instaladas. Que una bomba se vaya a romper depende de muchas condiciones; localización geográfica en la que se instale, cantidad de personas que van a usar esa bomba, calidad del agua, etc. Una vez se haya analizado el conjunto de datos, se procederá a realizar una configuración en los algoritmos y un preprocesado, no muy complejo, sobre el conjunto de datos para comprobar si se obtiene mayor precisión.

Los clasificadores elegidos son C4.5, Random Forest, Gradient Boosted, Fuzzy Rules, SVM, Naïve Bayes y red neuronal. He elegido 7 algoritmos ya que quiero estudiar cómo actúan los diferentes algoritmos (reglas, árboles de decisión, modelo probabilístico, etc.) sobre un mismo problema. Además, quiero saber cuál funciona mejor sobre un problema de clasificación con tres posibles clases.

Antes de continuar, añadir que algunos algoritmos necesitan un tratamiento básico sobre *missing values*, normalización o convertir variables categóricas a numéricas. Por tanto, la primera ejecución se realizará el tratamiento mínimo necesario sobre los modelos para que puedan ser ejecutados.

2. RESULTADOS OBTENIDOS

2.1. C4.5

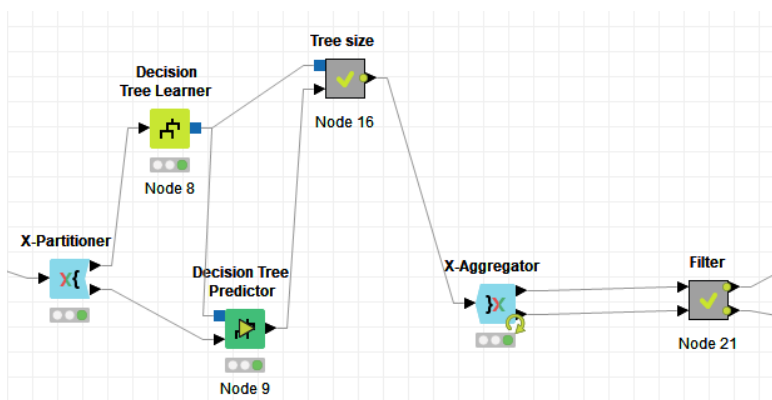


Figura 2.1: Flow de C4.5 en KNIME.

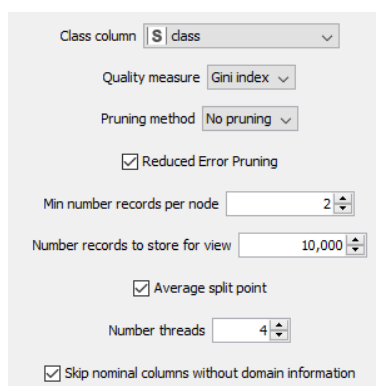


Figura 2.2: Configuración básica de C4.5 en KNIME.

Más adelante, vamos a cambiar parámetros en la configuración de C4.5 y vamos a estudiar cómo actúa el algoritmo con esos cambios comparándolo con su versión simple.

row ID	functional	non functional	functional needs repair
functional	26357	4223	1417
non functional	4593	17441	583
functional needs repair	2032	743	1497

Tabla 2.1: Matriz de confusión de C4.5.

Los resultados obtenidos son:

row ID	TPR	TNR	F1-score	G-mean	Accuracy
functional	0.823	0.753	0.811	0.787	0.769
non functional	0.771	0.863	0.774	0.815	
functional needs repair	0.350	0.963	0.385	0.581	
Overall					

Tabla 2.2: Estadísticas de C4.5.

La complejidad del modelo es el número de hojas. A través del nodo *Decision Tree to Ruleset*, se ve que tiene 6375 hojas.

2.2. RANDOM FOREST

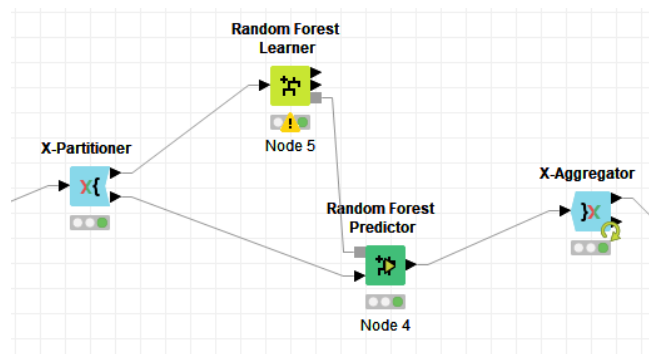


Figura 2.3: Flow de Random Forest en KNIME.

The image shows the 'Random Forest' configuration dialog in KNIME. Under 'Tree Options', the 'Split Criterion' is 'Information Gain Ratio'. The 'Limit number of levels (tree depth)' is set to 10, and the 'Minimum child node size' is set to 1. Under 'Forest Options', the 'Number of models' is set to 100, and the 'Use static random seed' checkbox is checked with a seed value of 123456.

Figura 2.4: Configuración básica de Random Forest en KNIME.

Con la configuración vista en la [Figura 2.4](#), se crean 100 modelos sobre 39 atributos. Por tanto, la complejidad de este algoritmo es esa, 100 modelos diferentes. Más adelante, cambiaremos estos parámetros para ver cómo actúa el algoritmo con nuevos parámetros.

row ID	functional	non functional	functional needs repair
functional	29472	2255	532
non functional	5457	17098	269
functional needs repair	2490	591	1236

Tabla 2.3: Matriz de confusión de Random Forest.

Los resultados obtenidos son:

row ID	TPR	TNR	F1-score	G-mean	Accuracy
functional	0.913	0.707	0.845	0.803	0.804
non functional	0.749	0.922	0.799	0.831	
functional needs repair	0.286	0.985	0.389	0.531	
Overall					

Tabla 2.4: Estadísticas de Random Forest.

2.3. GRADIENT BOOSTED

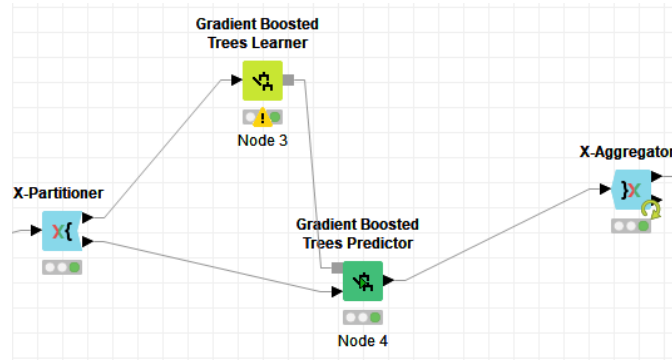


Figura 2.5: Flow de Gradient Boosted en KNIME.

Boosting Options	
Number of models	100
Learning rate	0.1

Figura 2.6: Configuración básica de Gradient Boosted en KNIME.

Con la configuración vista en la [Figura 2.6](#), se crean 100 modelos sobre 39 atributos con un learning rate muy bajito. La complejidad del algoritmo son 100 modelos.

row ID	functional	non functional	functional needs repair
functional	27125	3692	1442
non functional	4396	17824	604
functional needs repair	1998	714	1605

Tabla 2.5: Matriz de confusión de Gradient Boosted.

Los resultados obtenidos son:

row ID	TPR	TNR	F1-score	G-mean	Accuracy
functional	0.840	0.764	0.824	0.801	0.783
non functional	0.780	0.879	0.791	0.828	
functional needs repair	0.371	0.962	0.402	0.598	
Overall					

Tabla 2.6: Estadísticas de Gradient Boosted.

2.4. FUZZY RULES

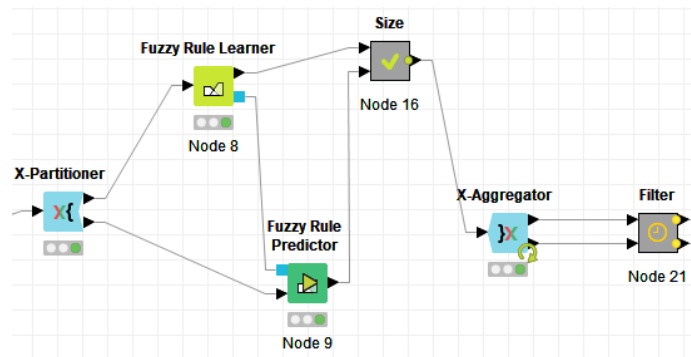


Figura 2.7: Flow de Gradient Boosted en KNIME.

row ID	functional	non functional	functional needs repair
functional	22850	5464	1173
non functional	7280	12841	692
functional needs repair	2066	866	979

Tabla 2.7: Matriz de confusión de Fuzzy Rules.

Los resultados obtenidos son:

row ID	TPR	TNR	F1-score	G-mean	Accuracy
functional	0.774	0.621	0.740	0.694	0.676
non functional	0.616	0.810	0.642	0.707	
functional needs repair	0.250	0.962	0.289	0.490	
Overall					

Tabla 2.8: Estadísticas de Fuzzy Rules.

La complejidad del modelo de Fuzzy Rules se ha calculado al igual que la complejidad de C4.5 pero sin realizar el paso de convertir el árbol de decisión en reglas, ya que Fuzzy Rules son reglas directamente. Por tanto, la complejidad de Fuzzy Rules son 14813 reglas.

2.5. SVM

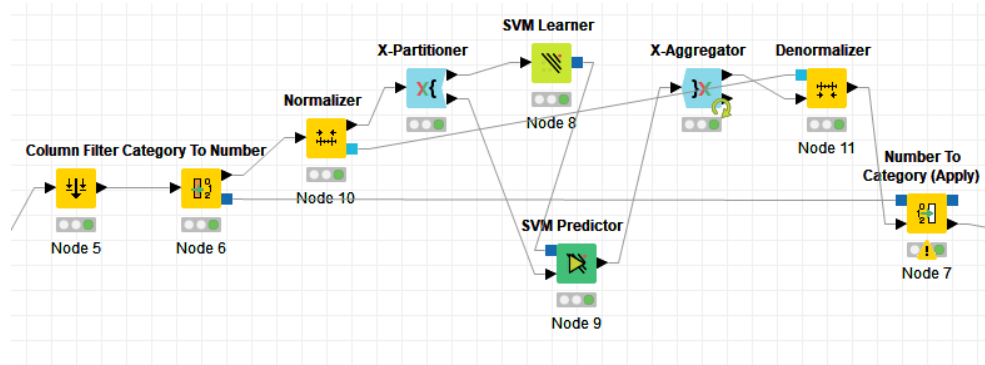


Figura 2.8: Flow de SVM en KNIME.

Para que la ejecución sea lo más ligera posible, elimino, a través de *Column filter*, algunas columnas con muchos valores diferentes. Además, transformo las variables categóricas en numéricas y las normalizo para que SVM pueda ejecutarse ya que requiere que los datos sean linealmente separables. Se establece la siguiente configuración:

Figura 2.9: Configuración básica de SVM en KNIME.

row ID	functional	non functional	functional needs repair
functional	23832	3549	0
non functional	10298	8493	0
functional needs repair	2991	678	0

Tabla 2.9: Matriz de confusión de SVM.

Los resultados obtenidos son:

row ID	TPR	TNR	F1-score	G-mean	Accuracy
functional	0.870	0.408	0.738	0.596	
non functional	0.451	0.863	0.539	0.624	
functional needs repair	0	1		0	
Overall					0.648

Tabla 2.10: Estadísticas de SVM.

Como dato curioso a añadir, este algoritmo estuvo más de 8 horas, una noche entera, hasta poder acabar. Para mi sorpresa, aún tardando tantas horas, no colgó KNIME y podía seguir usando mi ordenador mientras el algoritmo ejecutaba. Cuando finalizó la ejecución, observé que la clase minoritaria la toma como si no existiese a la hora de dividir las tres clases en un hiperplano. SVM no es capaz de dividir tres clases, lo cual es bastante curioso.

2.6. NAÏVE BAYES

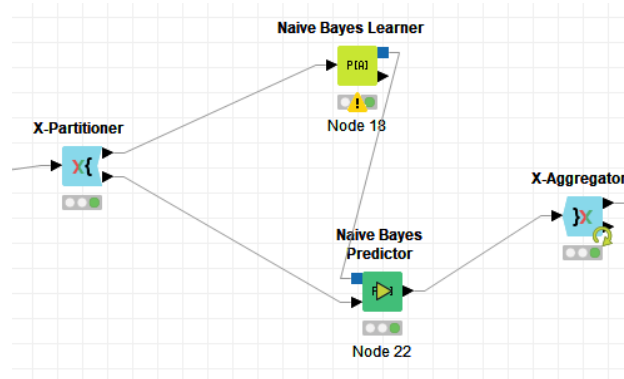


Figura 2.10: Flow de Naïve Bayes en KNIME.

Como se puede ver en la [Figura 2.10](#), no se ha realizado ningún preprocesado básico de datos. Por tanto, aparece un warning en el *Learner* que avisa de que se han ignorado varias columnas por tener muchos valores. Se establece la siguiente configuración:

Figura 2.11: Configuración básica de Naïve Bayes en KNIME.

row ID	functional	non functional	functional needs repair
functional	18933	6872	6454
non functional	4295	15492	3037
functional needs repair	848	909	2560

Tabla 2.11: Matriz de confusión de Naïve Bayes.

Los resultados obtenidos son:

row ID	TPR	TNR	F1-score	G-mean	Accuracy
functional	0.586	0.810	0.672	0.689	0.622
non functional	0.678	0.787	0.672	0.731	
functional needs repair	0.593	0.827	0.312	0.700	
Overall					

Tabla 2.12: Estadísticas de Naïve Bayes.

2.7. RProp MLP

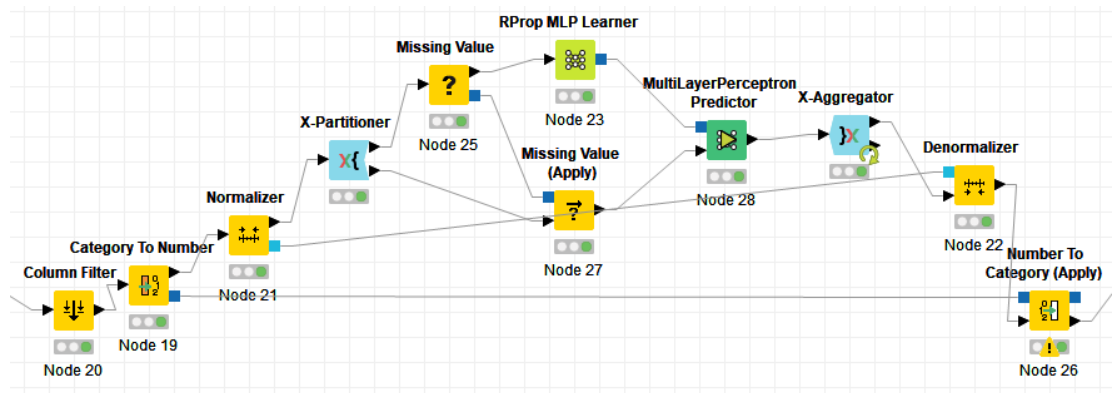


Figura 2.12: Flow de MLP en KNIME.

Por definición, las redes neuronales necesitan valores numéricos y normalizados. Además, se realiza un tratamiento sobre *missing values*. La configuración establecida es la siguiente:

Figura 2.13: Configuración básica de MLP en KNIME.

La complejidad de la red neuronal se puede observar en su configuración (Figura 2.13). Su configuración son 100 modelos, con una capa y 10 neuronas por capa. Este algoritmo va a ser tratado con una configuración distinta, centrándonos en el número de modelos, capas ocultas y neuronas, y un preprocesado posteriormente.

row ID	functional	non functional	functional needs repair
functional	27003	5173	83
non functional	9105	13639	80
functional needs repair	3152	1007	158

Tabla 2.13: Matriz de confusión de MLP.

Los resultados obtenidos son:

row ID	TPR	TNR	F1-score	G-mean	Accuracy
functional	0.837	0.548	0.755	0.677	0.686
non functional	0.597	0.831	0.639	0.704	
functional needs repair	0.036	0.990	0.068	0.191	
Overall					

Tabla 2.14: Estadísticas de MLP.

3. ANÁLISIS DE RESULTADOS

Decidí colocar todas las curvas ROC sobre la misma gráfica para que podamos ver todas las curvas de un vistazo y así facilitar el análisis visual.

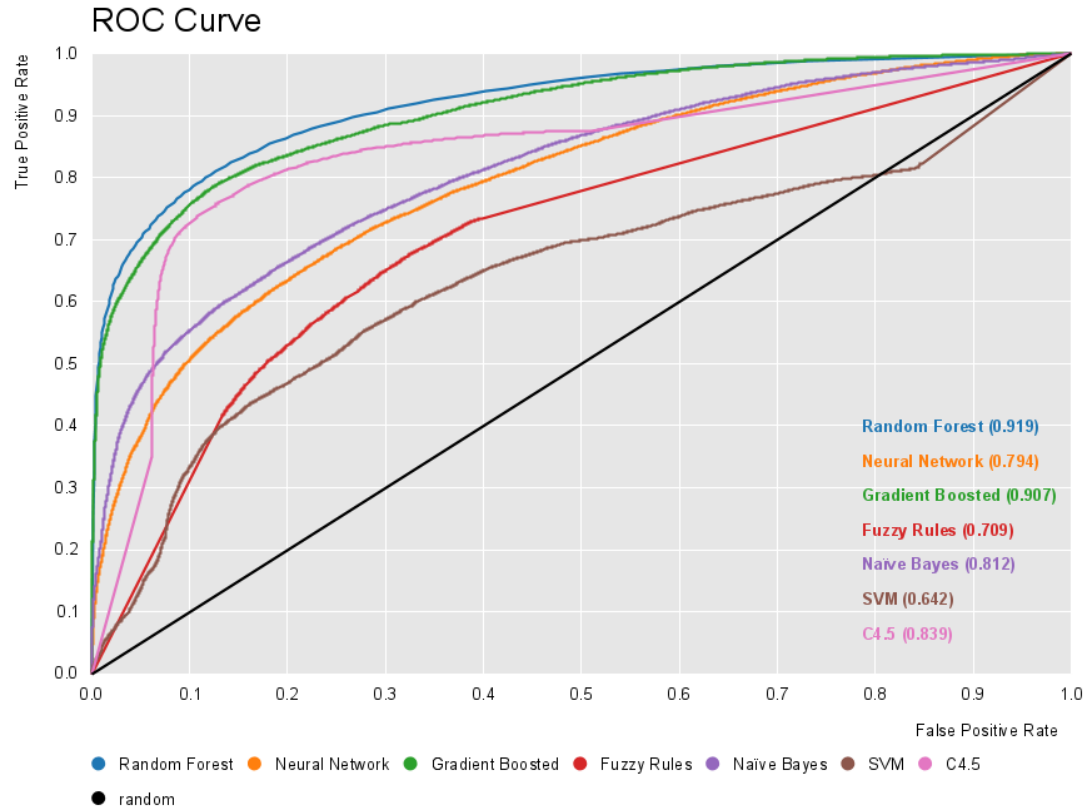


Figura 3.1: Curvas ROC de todos los algoritmos.

También se han agrupado todos los datos de los algoritmos en una misma tabla además de la incorporación de la columna *Area Under the Curve (AUC)*.

row ID	TPR	TNR	Accuracy	F1-score	G-mean	AUC
simple Random Forest	0.749	0.922	0.855	0.799	0.831	0.919
simple Neural Network	0.597	0.831	0.741	0.639	0.704	0.793
simple Gradient Boosted Trees	0.780	0.879	0.841	0.791	0.828	0.907
simple Fuzzy Rules	0.616	0.810	0.736	0.642	0.707	0.709
simple Naïve Bayes	0.678	0.787	0.745	0.672	0.731	0.811
simple SVM	0.451	0.863	0.708	0.539	0.624	0.641
simple C4.5	0.771	0.863	0.827	0.774	0.815	0.839

Tabla 3.1: Estadísticas de todos los algoritmos además del AUC.

He de resaltar que la columna *Accuracy*, en este caso, es la precisión sobre nuestra clase positiva. Es decir, la clase *non functional*.

Como se puede observar, los mejores resultados, tanto en precisión como en AUC, son Random Forest, Gradient Boosted Trees y C4.5. Pese a ser más robustas que los árboles de decisión por los pesos y su gran robustez frente al ruido, la red neuronal no está ofreciendo los resultados que podríamos esperar. Por tanto, intentaré sacarle mejor precisión al algoritmo en etapas posteriores.

Random Forest y Gradient Boosted es normal que arrojen buenos resultados ya que Random Forest es muy robusto al ruido y a los valores perdidos y Gradient Boosted, según podemos leer en [2] es útil en problemas con desbalanceo de clases, ya que aumenta el impacto de la clase positiva. En la siguiente tabla podemos ver el desbalanceo de clases:

row ID	Count (class)	Relative Frequency (class)
functional	32259	0.543
non functional	22824	0.384
functional needs repair	4317	0.072

Tabla 3.2: Distribución de las clases en nuestro problema.

Fuzzy Rules, Naïve Bayes y SVM son los que devuelven peor resultado debido al desbalanceo de clases. SVM se ve penalizado en

AUC al haber ignorado la clase minoritaria, *functional needs repair*, aunque en precisión no es tremendamente malo. Fuzzy Rules no está mal del todo pero la complejidad del modelo es bastante alta, por tanto es una elección cuestionable. Sobre Naïve Bayes, se intentará sacar mejor precisión en pasos posteriores con un preprocesado de datos, al igual que con C4.5.

4. CONFIGURACIÓN DE ALGORITMOS

4.1. C4.5

Las modificaciones realizadas sobre C4.5 son dos principalmente: Gain Ratio y C4.5 con poda.

4.1.1. C4.5 GAIN RATIO

En la configuración básica, la medida de calidad del algoritmo era Gini Index pero quería probar con la otra medida de calidad disponible. La configuración establecida es la siguiente:

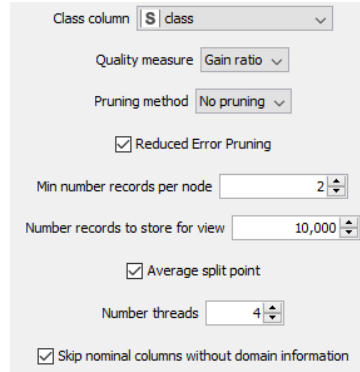
A screenshot of a software configuration window for the C4.5 algorithm. The window has a light gray background and contains several settings. At the top, 'Class column' is set to 'S | class' in a dropdown menu. Below it, 'Quality measure' is set to 'Gain ratio' in a dropdown menu. 'Pruning method' is set to 'No pruning' in a dropdown menu. There is a checked checkbox for 'Reduced Error Pruning'. Below that, 'Min number records per node' is set to 2 in a text box with up/down arrows. 'Number records to store for view' is set to 10,000 in a text box with up/down arrows. There is a checked checkbox for 'Average split point'. 'Number threads' is set to 4 in a text box with up/down arrows. At the bottom, there is a checked checkbox for 'Skip nominal columns without domain information'.

Figura 4.1: Configuración con Gain Ratio de C4.5.

4.1.2. C4.5 CON PODA

Con esta configuración, quería comprobar si se puede mantener una buena precisión reduciendo la complejidad del modelo y, por tanto, mejorando su interpretabilidad. La configuración era la siguiente:

The image shows a configuration window for the C4.5 decision tree algorithm. It includes several dropdown menus and checkboxes. The 'Class column' is set to 'S class'. The 'Quality measure' is set to 'Gini index'. The 'Pruning method' is set to 'MDL'. There are checkboxes for 'Reduced Error Pruning' (checked), 'Average split point' (checked), and 'Skip nominal columns without domain information' (checked). There are also input fields for 'Min number records per node' (set to 2), 'Number records to store for view' (set to 10,000), and 'Number threads' (set to 4).

Figura 4.2: Configuración de C4.5 con poda.

La tabla de resultados comparando las nuevas configuraciones con la simple es la siguiente¹:

row ID	TPR	TNR	Accuracy	F1-score	G-mean	AUC	Model size
simple C4.5	0.771	0.863	0.827	0.774	0.815	0.839	6375
C4.5 w/ pruning	0.708	0.899	0.826	0.758	0.798	0.870	1046.2
C4.5 GainRatio	0.737	0.867	0.817	0.756	0.799	0.828	6103.4
Preprocessed	0.754	0.860	0.819	0.763	0.805	0.826	8767.4

Tabla 4.1: Tabla de resultados de todas las ejecuciones de C45.

Como podemos ver en la [Tabla 4.1](#), las configuraciones con más precisión son la configuración básica, *Gini Index*, y la configuración con poda. Lo mismo pasa con *AUC*, además de que la configuración con poda destaca respecto a la configuración simple. Además, al haber realizado poda, el número de reglas de la configuración con poda es bastante inferior al resto, aumentando así su interpretabilidad.

Por tanto, podemos ver como la mejor opción para abordar este problema con C4.5 es realizando poda, mantenemos precisión, un buen *AUC* y buena interpretabilidad teniendo en cuenta el número de instancias del conjunto de datos.

Añado, a continuación, la curva ROC para visualizar el *AUC* mejor.

¹Nótese que aparece C4.5 preprocesado. Esa configuración la abordaremos más adelante aunque aparezca ya.

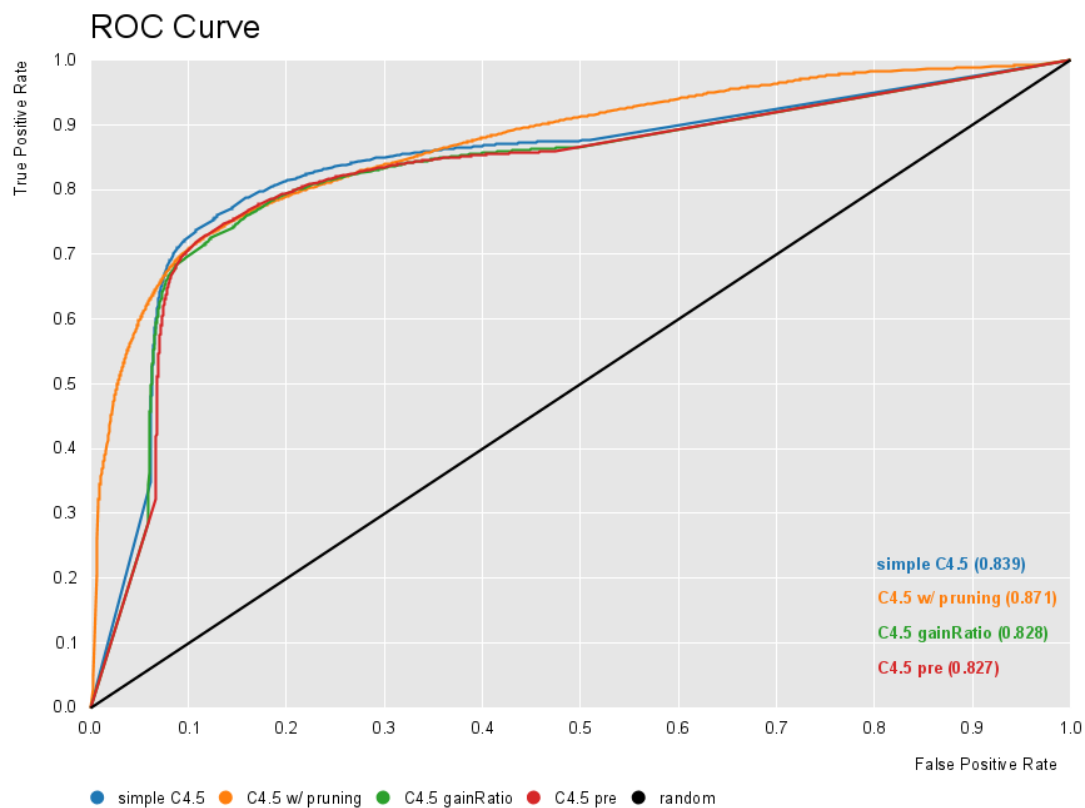


Figura 4.3: Curvas ROC de todas las configuraciones de C4.5.

4.2. RANDOM FOREST

Las modificaciones realizadas sobre Random Forest son: Gini Index y 200 modelos con Gain Ratio.

4.2.1. RANDOM FOREST GINI INDEX

En la configuración básica, la medida de calidad del algoritmo era Information Gain Ratio pero quería probar con la otra medida de calidad disponible. Además, quería ver qué modelo da mejores prestaciones ya que C4.5 tiene las mismas medidas de calidad. La configuración establecida es la siguiente:

The screenshot shows the 'Tree Options' section with 'Split Criterion' set to 'Gini Index'. Below it, 'Limit number of levels (tree depth)' is set to 10 and 'Minimum child node size' is set to 1. The 'Forest Options' section shows 'Number of models' set to 100. At the bottom, 'Use static random seed' is checked with the value 123456 and a 'New' button.

Figura 4.4: Configuración con Gini de Random Forest.

4.2.2. RANDOM FOREST 200 MODELOS

Con esta configuración, quería ver si duplicando el número de modelos que va a utilizar puede mejorar significativamente la precisión o el AUC . La configuración es la siguiente:

The screenshot shows the 'Tree Options' section with 'Split Criterion' set to 'Information Gain Ratio'. Below it, 'Limit number of levels (tree depth)' is set to 10 and 'Minimum child node size' is set to 1. The 'Forest Options' section shows 'Number of models' set to 200. At the bottom, 'Use static random seed' is checked with the value 123456 and a 'New' button.

Figura 4.5: Configuración de Random Forest con 200 modelos.

La tabla de resultados comparando las nuevas configuraciones con la simple es la siguiente²:

row ID	TPR	TNR	Accuracy	F1-score	G-mean	AUC
simple RForest	0.749	0.922	0.855	0.799	0.831	0.919
RForest Gini	0.75	0.922	0.855	0.800	0.831	0.920
RForest 200	0.755	0.921	0.857	0.803	0.834	0.920
Preprocessed	0.750	0.921	0.855	0.799	0.831	0.913

Tabla 4.2: Tabla de resultados de todas las ejecuciones de Random Forest.

²Nótese que aparece Random Forest preprocesado. Esa configuración la abordaremos más adelante aunque aparezca ya.

Como podemos ver en la [Tabla 4.2](#), la configuración con más precisión es la que utiliza 200 modelos para clasificar nuestro problema. Aún así, la diferencia respecto al resto de algoritmos es extremadamente baja. Esto me hace pensar que, aunque pongamos 500 modelos, muy difícilmente se va a mejorar significativamente la precisión con Random Forest.

Por tanto, a la hora de aplicar Random Forest, se puede aplicar cualquier configuración de las utilizadas y va a devolver resultados muy similares. Al menos en este problema.

Realizando una comparativa con C4.5, podemos ver en la [Tabla 4.1](#), como C4.5 simple y C4.5 Gain Ratio tienen menor precisión y *AUC* que utilizando Random Forest con Gini Index y Gain Ratio. Lo cual es normal, pues Random Forest utiliza 100 clasificadores diferentes.

Añado, a continuación, la curva ROC para visualizar el AUC mejor.

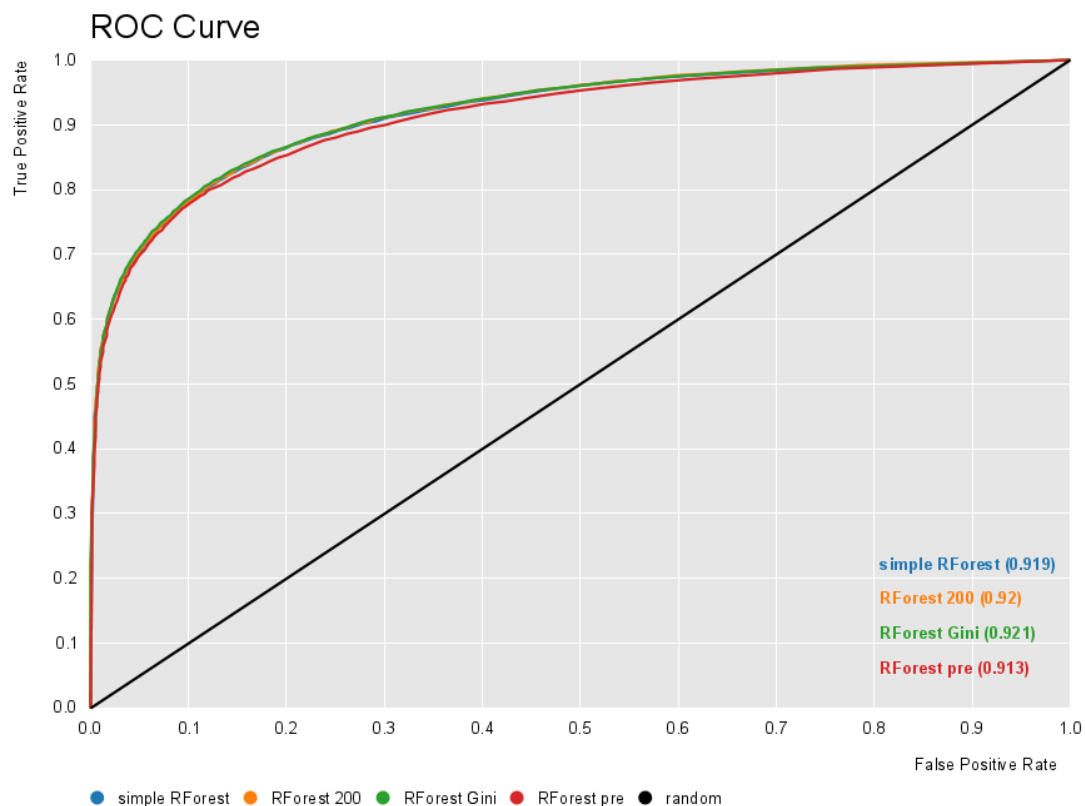


Figura 4.6: Curvas ROC de todas las configuraciones de Random Forest.

4.3. RPROP MLP

Las modificaciones realizadas sobre la red neuronal son ejecutarlo con 200 modelos pero diferenciando en que una opción tiene 3 capas y 100 neuronas, mientras que la otra tiene 6 capas y 50 neuronas. Con este experimento, quiero comprobar qué elemento es más relevante a la hora de obtener mejor precisión, si las capas ocultas o las neuronas.

4.3.1. MLP CON 200 MODELOS, 3 CAPAS OCULTAS Y 100 NEURONAS POR CAPA.

La configuración establecida es la siguiente:

Maximum number of iterations: 200

Number of hidden layers: 3

Number of hidden neurons per layer: 100

class column: S | class

☐ Ignore Missing Values

☐ Use seed for random initialization

Random seed: 123,456

Figura 4.7: Configuración de red neuronal con 200 modelos, 3 capas ocultas y 100 neuronas por capa.

4.3.2. MLP CON 200 MODELOS, 6 CAPAS OCULTAS Y 50 NEURONAS POR CAPA.

La configuración es la siguiente:

Maximum number of iterations: 200

Number of hidden layers: 6

Number of hidden neurons per layer: 50

class column: S | class

☐ Ignore Missing Values

☐ Use seed for random initialization

Random seed: 123,456

Figura 4.8: Configuración de red neuronal con 200 modelos, 6 capas ocultas y 50 neuronas por capa.

La tabla de resultados comparando las nuevas configuraciones con la simple es la siguiente³:

³Nótese que aparece MLP preprocesado. Esa configuración la abordaremos más adelante aunque aparezca ya.

row ID	TPR	TNR	Accuracy	F1-score	G-mean	AUC
100/1/10 NNetwork	0.597	0.831	0.741	0.639	0.704	0.793
200/6/50 NNetwork	0.648	0.891	0.798	0.711	0.760	0.848
200/3/100 NNetwork	0.678	0.887	0.806	0.729	0.775	0.862
NNetwork pre	0.595	0.784	0.711	0.613	0.683	0.754

Tabla 4.3: Tabla de resultados de todas las ejecuciones de la red neuronal.

Como podemos ver en la [Tabla 4.3](#), la configuración con más precisión es la que utiliza 3 capas ocultas y 100 neuronas por capa. Pese a que nuestro problema no es linealmente separable, un exceso de capas ocultas, en este caso 6, no proporciona mejores resultados que 3 capas y 100 neuronas. Esto me hace pensar que 3 capas ocultas son suficientes para estudiar este problema y las 100 neuronas apoyan para obtener buenos resultados. Aún así, la diferencia respecto a la otra opción en términos de precisión no es muy grande mientras que en *AUC* es algo superior. Como era de esperar, la configuración simple de 100 modelos, 1 capa oculta y 10 neuronas por capa se queda muy por detrás respecto a las otras configuraciones.

Como comenté en el apartado anterior, las redes neuronales deberían proporcionar mejores resultados que los árboles de decisión. Si comparamos con la [Tabla 4.1](#), vemos que la configuración de 3 capas ocultas y 100 neuronas sigue ofreciendo peor precisión que la configuración simple de C4.5 pero mejor valor *AUC*. Si ahora comparamos con C4.5 con poda, la red neuronal se queda muy por detrás en ambos valores. Esto me hace pensar que se podría encontrar mejor configuración de la red neuronal a base de prueba y error.

Añado, a continuación, la curva ROC para visualizar el *AUC* mejor.

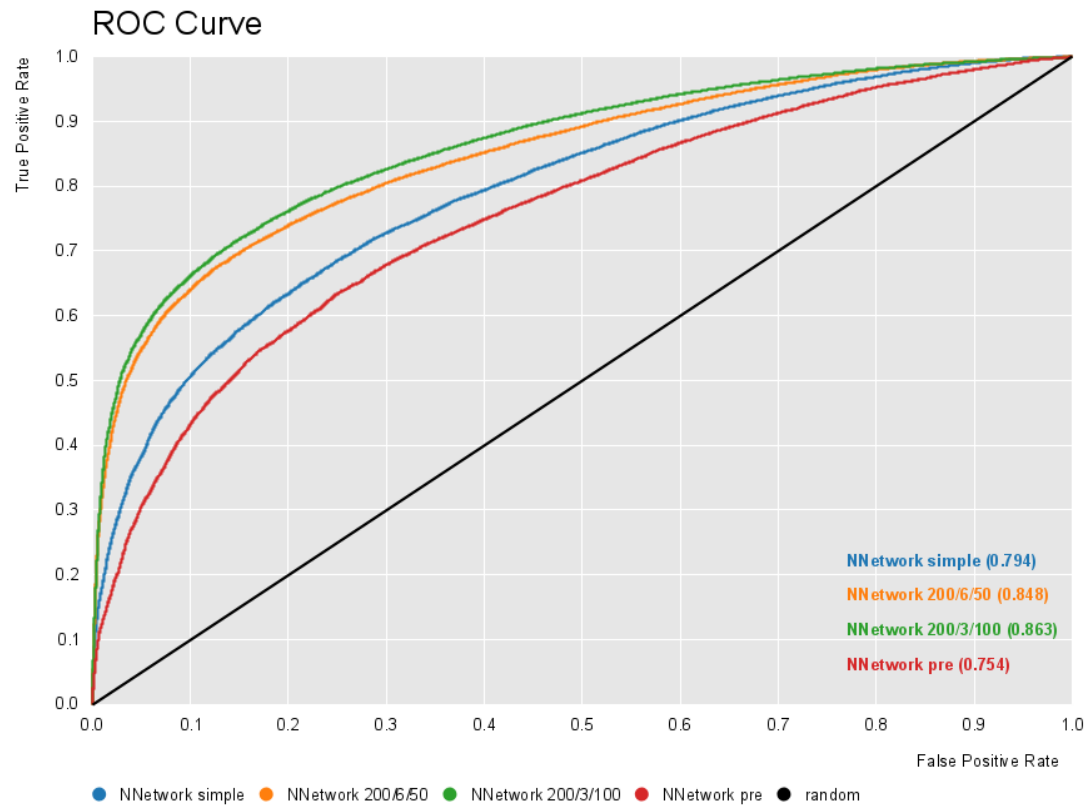


Figura 4.9: Curvas ROC de todas las configuraciones de la red neuronal.

5. PROCESADO DE DATOS

El proceso de preprocesado de los datos que he seguido es la siguiente:

1. He filtrado varios atributos a través del nodo *Column Filter*. Los atributos seleccionados para excluirlos del conjunto de datos han sido estudiado previamente a través del nodo *Linear Correlation* ya que existían diferentes atributos que reflejen la misma información. Además de esos atributos, he seleccionado ciertos atributos, debido a mi análisis sobre el conjunto de datos y qué era cada atributo, para excluirlos también ya que no eran atributos de calidad (*date_recorded*, *basin*, *region*, *region_code*, etc).

2. Una vez filtrado los atributos, se imputan *missing values* de forma sencilla a través del nodo *Missing Values*.
3. Por último, he realizado un balanceo de clases para equilibrar las clases a través del algoritmo *Smote*.

La distribución de las clases previa a la ejecución del algoritmo Smote es la siguiente:

row ID	Count (class)	Relative Frequency (class)
functional	25808	0.543
non functional	18259	0.384
functional needs repair	3453	0.072

Tabla 5.1: Distribución de clases antes del algoritmo Smote.

Y la distribución de las clases posterior a la ejecución es:

row ID	Count (class)	Relative Frequency (class)
functional	25808	0.333
functional needs repair	25808	0.333
non functional	25808	0.333

Tabla 5.2: Distribución de clases después del algoritmo Smote.

Antes de comenzar con el análisis, señalar que las ejecuciones con preprocesado de datos tienen la misma configuración que las configuraciones básicas de cada algoritmo.

5.1. C4.5

row ID	TPR	TNR	Accuracy	F1-score	G-mean	AUC	Model size
simple C4.5	0.771	0.863	0.827	0.774	0.815	0.839	6375
C4.5 w/ pruning	0.708	0.899	0.826	0.758	0.798	0.870	1046.2
C4.5 GainRatio	0.737	0.867	0.817	0.756	0.799	0.828	6103.4
Preprocessed	0.754	0.860	0.819	0.763	0.805	0.826	8767.4

Tabla 5.3: Tabla de resultados de todas las ejecuciones de C45.

C4.5 al realizar una exclusión de atributos, imputación de valores perdidos y un balanceo de clases proporciona peores resultados que si no se hiciese nada más que lo necesario para la ejecución. Su complejidad aumenta considerablemente, lo que es normal pues se ha aumentado muchísimo el número de instancias del conjunto de datos.

5.2. RANDOM FOREST

row ID	TPR	TNR	Accuracy	F1-score	G-mean	AUC
simple RForest	0.749	0.922	0.855	0.799	0.831	0.919
RForest Gini	0.75	0.922	0.855	0.800	0.831	0.920
RForest 200	0.755	0.921	0.857	0.803	0.834	0.920
Preprocessed	0.750	0.921	0.855	0.799	0.831	0.913

Tabla 5.4: Tabla de resultados de todas las ejecuciones de Random Forest.

Random Forest, tanto de forma simple como preprocesado, devuelve resultados muy similares. Esto es debido a que el Random Forest es muy robusto al ruido y no se ve penalizado por el desbalanceo de clases.

5.3. RPROP MLP

row ID	TPR	TNR	Accuracy	F1-score	G-mean	AUC
100/1/10 NNetwork	0.597	0.831	0.741	0.639	0.704	0.793
200/6/50 NNetwork	0.648	0.891	0.798	0.711	0.760	0.848
200/3/100 NNetwork	0.678	0.887	0.806	0.729	0.775	0.862
NNetwork pre	0.595	0.784	0.711	0.613	0.683	0.754

Tabla 5.5: Tabla de resultados de todas las ejecuciones de la red neuronal.

Observando los datos, podemos comprobar que el *TPR* se mantiene igual en la configuración básica como en el algoritmo preprocesado mientras que el *TNR*, el número de predicciones negativas correctas entre el total, disminuye considerablemente. Este es el motivo por

el que la precisión se ve reducida al igual que el AUC y el resto de medidas.

5.4. NAÏVE BAYES

row ID	TPR	TNR	Accuracy	F1-score	G-mean	AUC
simple Bayes	0.678	0.787	0.745	0.672	0.731	0.811
pre Bayes	0.679	0.707	0.697	0.633	0.693	0.772

Tabla 5.6: Tabla de resultados de las ejecuciones de Naïve Bayes.

Al igual que el algoritmo MLP, la precisión se ve afectada por el descenso en TNR , al igual que el resto de medidas.

6. INTERPRETRACIÓN DE RESULTADOS

Los modelos con peor rendimiento son Fuzzy Rules y SVM. Pese a que SVM ignoró la clase minoritaria, *Functional needs repair*, no ha logrado obtener buena precisión ni área sobre la curva. Por otro lado, uno de los modelos con peor rendimiento es Naïve Bayes pese a ser una buena opción para problemas con múltiples clases. Además, para mi sorpresa, realizando preprocesado proporciona peores resultados que si no lo haces.

Los mejores modelos han sido Random Forest, con su configuración de 200 modelos, y Gradient Boosted. Pese a que Gradient Boosted no recibió ninguna configuración específica, es capaz de mantenerse como el segundo modelo con mejor precisión y área sobre la curva. Esto me hace pensar que hubiera sido interesante aplicarle una configuración más avanzada, como 200 modelos, aunque podría haber sucedido el mismo suceso que con Random Forest. Es decir, que apenas mejorase ya que la diferencia entre 100 modelos y 200 modelos no es significativa.

Por otro lado, la red neuronal no es un mal modelo para ejecutar sobre este problema pero no es la mejor opción. Pese haber probado dos configuraciones distintas, no ha sido capaz de tener un rendi-

miento mejor que C4.5.

Como modelo más curioso me quedo con C4.5 en su configuración de poda pues ha conseguido mantener la precisión y *AUC* pero mejorando la interpretabilidad enormemente. La diferencia entre la complejidad de la configuración básica de C4.5 y la configuración con poda es de, más o menos, 5000 reglas. Me parece una total locura mantener buen rendimiento con tantísimas reglas menos.

Por tanto, después de haber estudiado los modelos con diferentes configuraciones y/o preprocesado básico de datos, concluyo que todos tienen un rendimiento adecuado, unos más que otros, frente a este problema.

7. BIBLIOGRAFÍA

REFERENCIAS

- [1] <https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial->
Consultado el 21 de Octubre.
- [2] <https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/>, Consultado el 28 de Octubre.
- [3] https://www.researchgate.net/post/Understanding_AUC_of_ROC_sensitivity_and_specificity_values, Consultado el 28 de Octubre.
- [4] http://rikunert.com/SMOTE_explained, Consultado el 2 de Octubre.