

# Práctica 1 de Inteligencia de Negocio



# Universidad de Granada

Antonio Manuel Fresneda Rodríguez

[antoniomfr@correo.ugr.es](mailto:antoniomfr@correo.ugr.es)

4 de noviembre de 2019

# Índice

<b>Introducción</b>	<b>3</b>
<b>Resultados obtenidos</b>	<b>3</b>
C4.5 . . . . .	4
Naive Bayes . . . . .	5
Random Forest . . . . .	6
MLP . . . . .	7
KNN . . . . .	8
XGBoosting . . . . .	9
<b>Análisis de resultados</b>	<b>10</b>
Clase Funcional . . . . .	10
Clase Funcional needs repair . . . . .	11
Clase non functional . . . . .	12
Sobre-aprendizaje . . . . .	12
<b>Interpretación de los resultados</b>	<b>15</b>
<b>Ejecución de algoritmos modificando parámetros</b>	<b>16</b>
Clase funcional . . . . .	16
Clase funcional needs repair . . . . .	17
Clase non functional . . . . .	18
Análisis . . . . .	19
<b>Procesado de datos</b>	<b>20</b>
Pasos seguidos . . . . .	20
Resultados . . . . .	22
Clase funcional . . . . .	22
Clase funcional needs repair . . . . .	23
Clase non functional . . . . .	24
Sobre-aprendizaje . . . . .	25
Análisis . . . . .	27

## Introducción

El problema que se esta abordando en esta práctica es el de detección de fallos de bombas de agua en Tanzania. El dataset se ha obtenido del siguiente [enlace](#).

El dataset contiene 40 atributos y 59400 instancias y tenemos 3 clases para clasificar: functional, functional needs repair y non functional.

Los seis algoritmos que se han usado han sido:

1. **C4.5:** Algoritmo con el que generamos árboles de decisión.
2. **Naive Bayes:** Clasificador probabilístico basado en el teorema de Bayes.
3. **Random Forest:** Combina varios árboles de decisión.
4. **Multilayer perceptron:** Red neuronal.
5. **KNN:** Clasificador del vecino más cercano basado en distancias.
6. **XGBoosting:** Algoritmo que usa varios clasificadores débiles que se combinan para dar una salida.

Se ha usado validación cruzada de 5 particiones fijando la semilla en 444555.

## Resultados obtenidos

Para medir el comportamiento de los algoritmos se han usado TPR, TNR F1-score AUC, etc y se ha realizado una tabla en la que se recogen dichas medidas por cada clase que debemos clasificar. Así podemos ver como de bien clasifica cada algoritmo cada una de las clases que tenemos que clasificar.

En cada apartado se explica la configuración de cada algoritmo y el porcentaje de acierto (casos acertados entre casos posibles) de cada algoritmo. En la sección final se expone una tabla con todas las medidas obtenidas de cada algoritmo para cada clase, junto a las curvas ROC correspondientes.

Para todos los algoritmos se ha valorado el sobre ajuste, ejecutando un predictor con los datos de entrenamiento, sacando la media geométrica de cada clase y haciendo un ratio entre las dos.

## C4.5

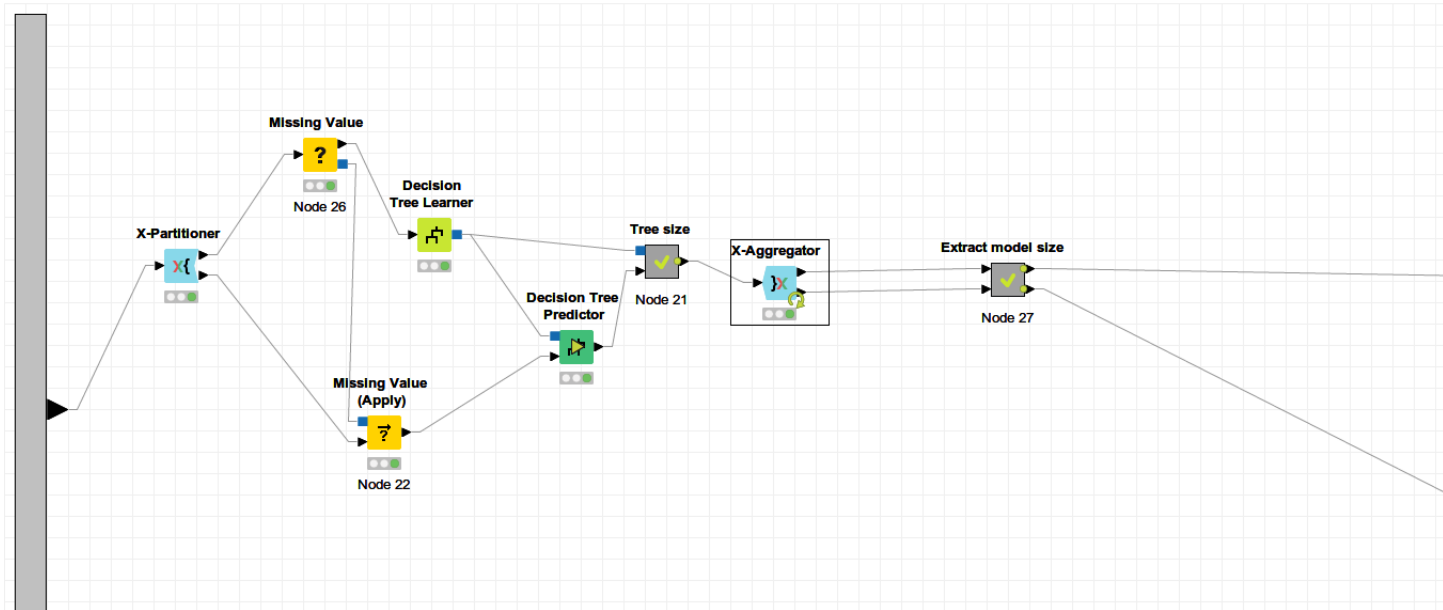


Figura 1: Captura de nodo C4,5

Para ejecutar el algoritmo se ha tenido que añadir valores perdidos, ya que este algoritmo no los acepta. Para ello se ha usado el nodo missing usando la media para valores numéricos y la moda para strings.

Los nodos tree size y model size se encargan de extraer las medidas de complejidad del modelo que se verán en la tabla de comparaciones.

Los valores se han dejado por defecto.

## Naive Bayes

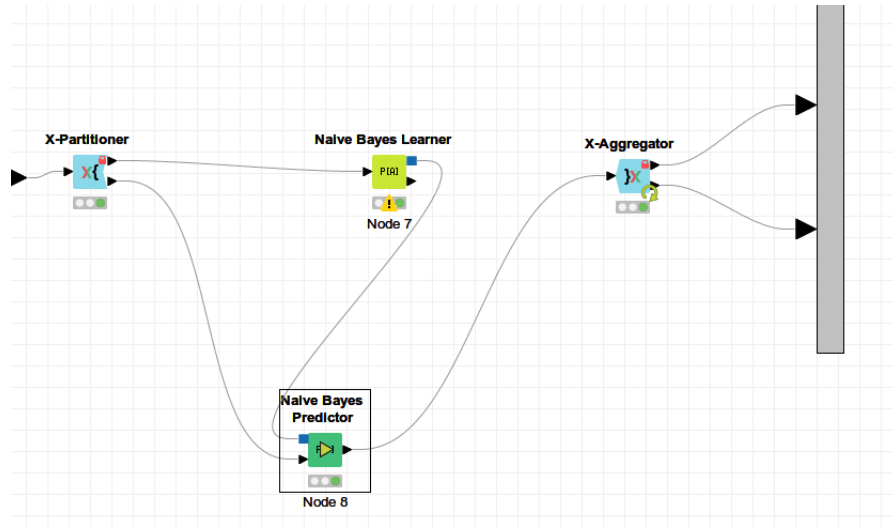


Figura 2: Captura de nodo Naive Bayes

Este algoritmo, a diferencia del anterior si es capaz de manejar valores perdidos.

El warning que aparece nos dice que algunas columnas se han omitido ya que tienen demasiados valores. Se ha probado a usar antes un nodo domain calculator, pero aún así este warning ha seguido apareciendo.

Los parámetros se han dejado por defecto.

## Random Forest

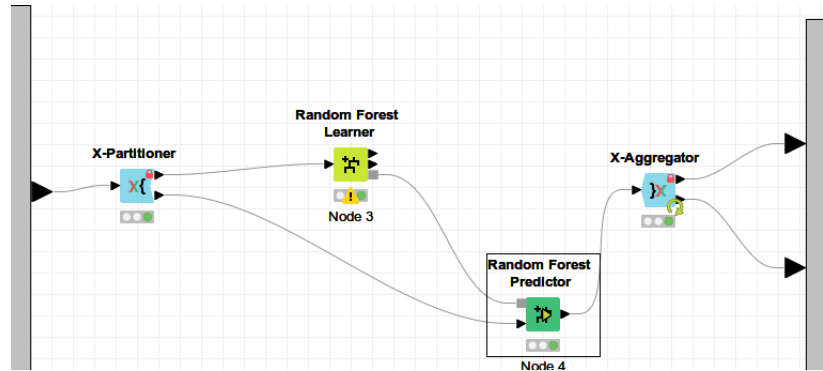


Figura 3: Captura de nodo Random Forest

Justo como el algoritmo anterior, el nodo Random Forest es capaz de manejar los valores perdidos y no ha sido necesario aplicar el nodo missing. El warning sigue siendo por culpa de que hay columnas con demasiados valores y se han omitido. Los parametros se han dejado por defecto.

## MLP

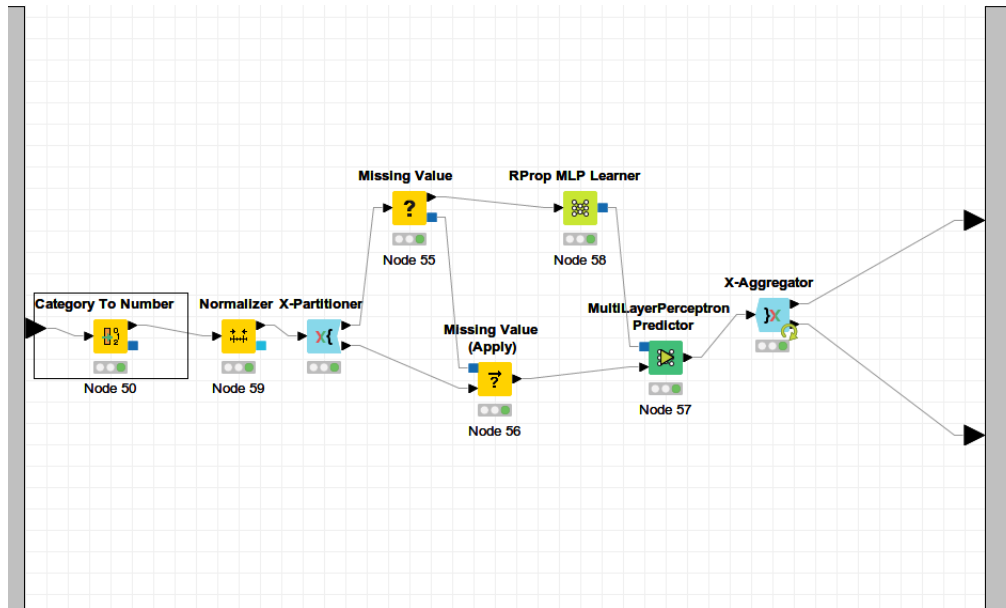


Figura 4: Captura de nodo MLP

Este algoritmo si ha necesitado algo más de procesado.

Primero hemos cambiado los valores categóricos por numéricos. Podríamos haber usado el nodo *one to many*, pero se ha optado por este nodo debido a que al haber columnas con tantos valores diferentes, se iban a añadir una gran cantidad de columnas que iban a comprometer el rendimiento (en tiempo) del algoritmo.

Seguidamente se han imputado valores perdidos en los que los atributos de tipo double se ha hecho la media y en los de tipo entero se ha usado la moda (ya que son los valores categóricos que se han cambiado por enteros y no tiene sentido calcular la media de estos valores).

Los parámetros se han dejado por defecto.

## KNN

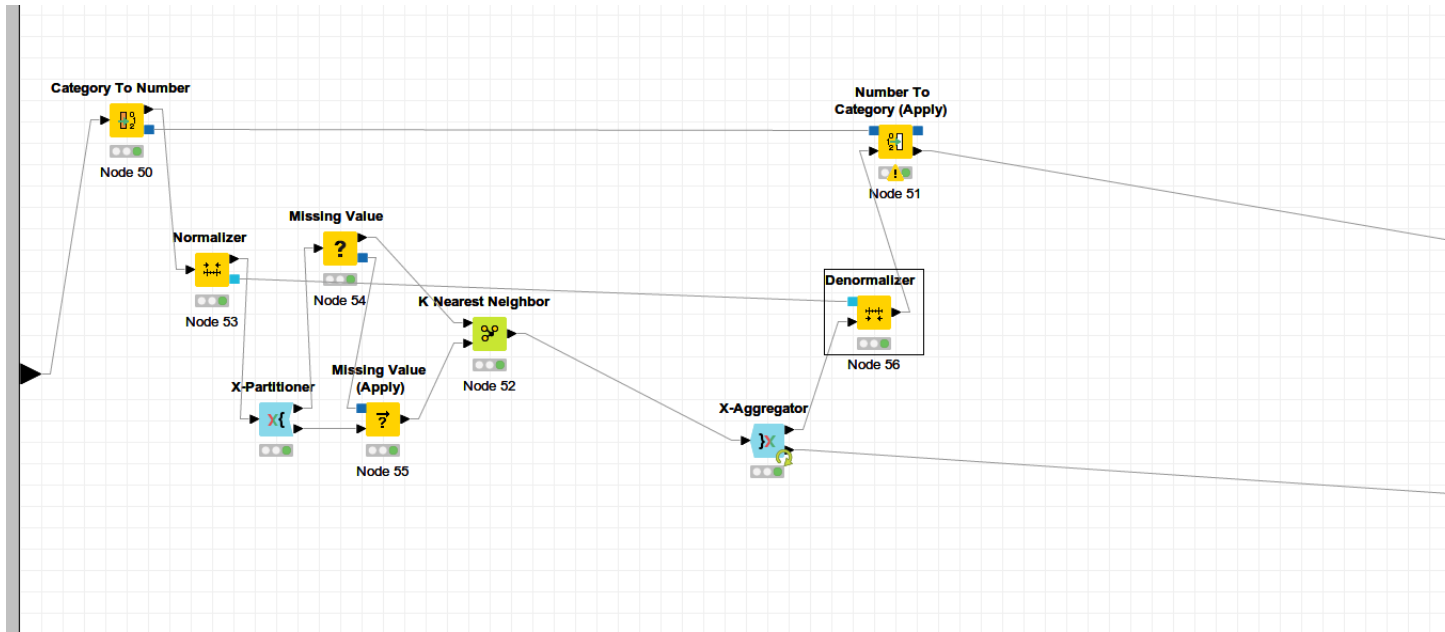


Figura 5: Captura de nodo KNN

Este ha sido otro algoritmo que ha necesitado algo más de procesado, ya que si queremos usar KNN necesitamos normalizar los datos.

Primero se han pasado las variables categóricas a numéricas (siguiendo el criterio anterior), se han normalizado los datos usando la normalización min-max.

Una vez los datos se han normalizado, se han imputado los valores perdidos, se ha ejecutado el algoritmo y se ha vuelto el dataset a su estado original, denormalizando los datos y volviendo las variables numéricas que se habían cambiado a categóricas de nuevo.

Los parámetros de configuración se han dejado por defecto.

## XGBoosting

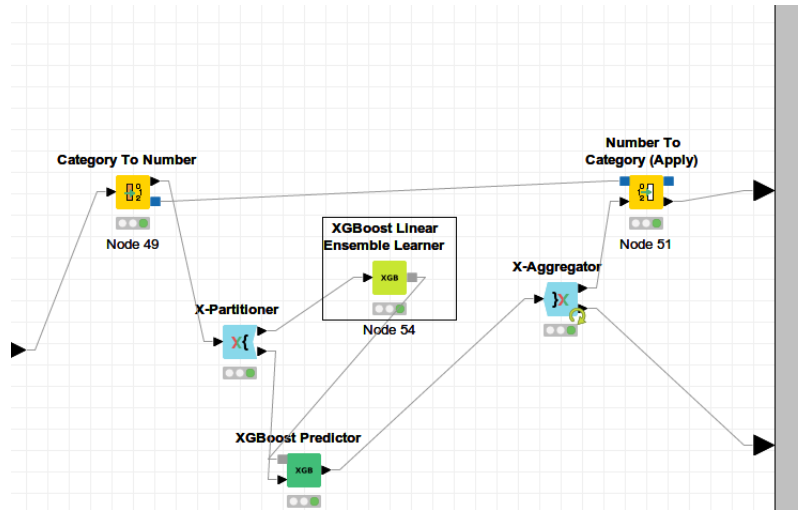


Figura 6: Captura de nodo XGBoosting

Este algoritmo por defecto maneja valores perdidos y no es necesario aplicar el nodo missing, solo se han cambiado las variables categóricas por numéricas. El resto de parámetros si se han dejado por defecto.

## Análisis de resultados

A continuación se muestran unas tablas donde se encuentran las medidas de acierto de los distintos algoritmos que se han usado y una gráfica donde se muestran las curvas ROC correspondientes a los algoritmos usados:

### Clase Functional

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-M	Rcll	Size	AUC
C4.5	26395	6615	20265	5584	0.83	0.75	0.80	0.79	0.81	0.79	0.83	6366.40	0.81
NB	19182	6253	20888	13077	0.59	0.77	0.75	0.67	0.66	0.68	0.59		0.77
RF	29509	7891	19250	2750	0.91	0.71	0.79	0.82	0.85	0.81	0.91		0.90
MLP	27116	12590	14551	5143	0.84	0.54	0.68	0.70	0.75	0.67	0.84		0.77
KNN	26869	8841	18300	5390	0.83	0.67	0.75	0.76	0.79	0.75	0.83		0.82
XGB	26635	14519	12622	5624	0.83	0.47	0.65	0.66	0.73	0.62	0.83		0.71

Cuadro 1: Tabla de resultados de clase funcional

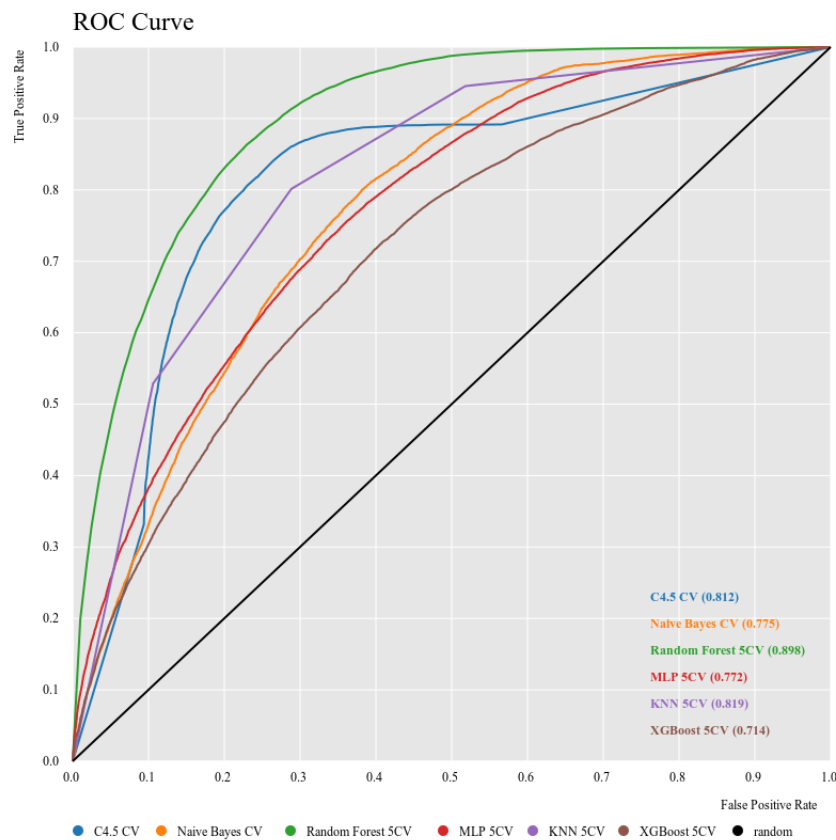


Figura 7: Curva ROC: Clase positiva: funcional

## Clase Functional needs repair

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-M	RcII	Size	AUC
C4.5	1468	1958	52612	2821	0.34	0.96	0.43	0.92	0.38	0.57	0.34	6366.40	0.73
NB	2197	9794	45289	2120	0.51	0.82	0.18	0.80	0.27	0.65	0.51		0.73
RF	1217	811	54272	3100	0.28	0.99	0.60	0.93	0.38	0.53	0.28		0.85
MLP	27	26	55057	4290	0.01	1.00	0.51	0.93	0.01	0.08	0.01		0.70
KNN	1204	1491	53592	3113	0.28	0.97	0.45	0.92	0.34	0.52	0.28		0.73
XGB	57	71	55012	4260	0.01	1.00	0.45	0.93	0.03	0.11	0.01		0.71

Cuadro 2: Tabla de resultados de clase functional needs repair

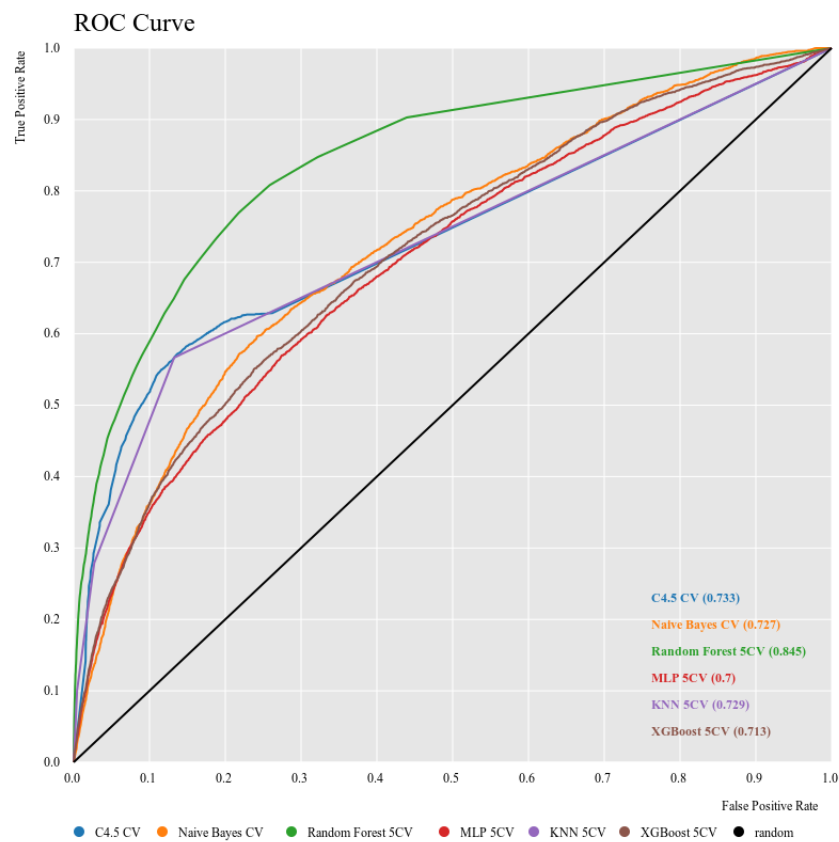


Figura 8: Curva ROC: Clase positiva: functional needs repair

## Clase non functional

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-M	Rcll	Size	AUC
C4.5	17462	4961	31307	5129	0.77	0.86	0.78	0.83	0.78	0.82	0.77	6366.40	0.84
NB	14716	7258	29318	8108	0.64	0.80	0.67	0.74	0.66	0.72	0.64		0.79
RF	17155	2817	33759	5669	0.75	0.92	0.86	0.86	0.80	0.83	0.75		0.92
MLP	13520	6121	30455	9304	0.59	0.83	0.69	0.74	0.64	0.70	0.59		0.79
KNN	15806	5189	31387	7018	0.69	0.86	0.75	0.79	0.72	0.77	0.69		0.84
XGB	11551	6567	30009	11273	0.51	0.82	0.64	0.70	0.56	0.64	0.51		0.73

Cuadro 3: Tabla de resultados de clase non funcional

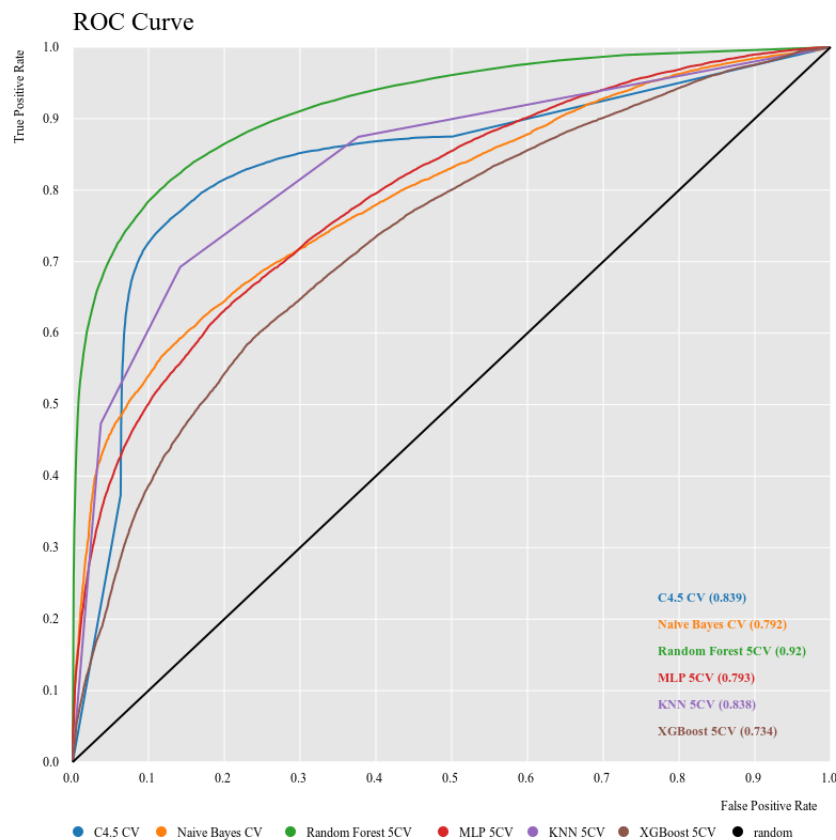


Figura 9: Curva ROC: Clase positiva: non funcional

## Sobre-aprendizaje

En esta primera sección se va a comentar si los distintos algoritmos que se han empleado han sufrido de sobre-aprendizaje o no.

Como se ha comentado previamente, para analizar el sobre-aprendizaje se ha calculado la media geométrica tanto en entrenamiento como en test y se ha calculado un ratio dividiendo el valor de G-mean en training por el valor de G-mean en test.

Se ha añadido a cada uno de los algoritmos un nodo que se encarga de calcular este ratio y los resultados son los siguientes:

■ **C4.5**

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.94	0.79	1.19
non functional	0.95	0.82	1.16
functional needs repair	0.82	0.57	1.44

■ **Naive Bayes**

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.68	0.68	1.00
non functional	0.72	0.72	1.00
functional needs repair	0.65	0.65	1.00

■ **Random Forest**

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.91	0.81	1.13
non functional	0.93	0.83	1.11
functional needs repair	0.75	0.53	1.42

### ■ Multilayer Perceptron

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.68	0.68	1.00
non functional	0.71	0.71	1.00
functional needs repair	0.14	0.13	1.05

### ■ KNN

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.85	0.75	1.14
non functional	0.88	0.77	1.14
functional needs repair	0.69	0.52	1.32

### ■ XGBoosting

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.62	0.62	1.00
non functional	0.65	0.64	1.00
functional needs repair	0.12	0.11	1.05

Estudiando las tables anteriores, podemos deducir que el algoritmo C4.5 si ha sufrido un poco de sobre-aprendizaje con la clase *functional needs repair*, ya que en training ha sacado un 82 % y en test un 53 %. Para intentar mejorar el funcionamiento deberíamos elegir un método de poda para el árbol.

## Interpretación de los resultados

Mirando los resultados obtenidos, vemos como el algoritmo que mejor se ha comportado ha sido Random Forest en cada una de las tres clases. Seguido de C4.5 y KNN.

Si el objetivo de estudiar este dataset es comprender el porqué una bomba puede fallar, claramente elegiríamos C4.5, aunque los árboles de decisión resultantes tienen más o menos unas 60.000 hojas, cosa que hace que el modelo sea bastante difícil de interpretar, pero aún así, ya es mucho más interpretable que una red neuronal o que XGBoosting.

Si lo que estamos buscando es porcentaje de acierto y no nos interesa la interpretabilidad, yo elegiría Random Forest, redes neuronales con mas capas ocultas (en teoría debería de mejorar el acierto a medida que vamos aumentando el número de capas ocultas teniendo cuidado con el overfitting) o KNN, ya que son modelos que tienen una capacidad mayor de extraer conocimiento del dataset a costa de tener una menor o nula interpretabilidad.

Aunque Knime permite tener una representación gráfica de todos los árboles que forman Random Forest, creo que sigue sin ser un modelo fácilmente interpretable, ya que tendríamos que estudiar cada uno de los árboles que forman Random Forest (en caso de Knime por defecto son 100).

En cuanto al Perceptron Multicapa, el resultado ha sido un poco decepcionante, ya que ha tenido un comportamiento parecido a algoritmos bastante más simples como Naive Bayes. La explicación a este comportamiento puede deberse a varios factores:

- Solo se ha ejecutado con una capa oculta de neuronas. Se ha ejecutado la red con dos capas ocultas. Los resultados están reflejados en la siguiente sección.
- Que la red haya sufrido de sobre-aprendizaje: Como podemos ver en la sección anterior, con los resultados de analizar el sobre-aprendizaje deducimos que en este caso la red no ha sufrido de sobre-aprendizaje, ya que el porcentaje de acierto tanto en training como en test es muy parecido.

Luego si queremos subir el acierto usando MLP deberíamos aumentar el número de capas ocultas de la red.

En cuanto al resto de algoritmos (Naive Bayes y XGBoosting) no proporcionan mejora en cuanto a interpretabilidad ni acierto así que desde mi punto de vista descartaría el uso de estos algoritmos, ya que hemos comprobado que hay otros que funcionan mejor en este dataset.

## Ejecución de algoritmos modificando parámetros

Como se ha comentado antes, los dos algoritmos que se van a modificar sus parámetros van a ser MLP (añadiendo una o más capas ocultas) y C4.5 añadiendo un método de poda. Para añadir un método de poda a C4.5 el nodo tree learner tiene una opción llamada *prunning method* en la que se usa una algoritmo de poda llamado MLD( minimum description length). Es un algoritmo que ayuda a conocer los atributos con mayor influencia sobre otro atributo objetivo.

Para aumentar el número de capas ocultas en MLP es fácil, simplemente tiene una opción en la que podemos aumentar el numero de capas ocultas.

Se han añadido tablas comparativas de estos dos algoritmos y Random Forest, ya que ha sido el algoritmo que mejor ha funcionado dejando parámetros por defecto y asi podemos ver si realmente la modificación de los parámetros ha supuesto una mejora significativa.

Los resultados son los siguientes:

### Clase functional

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-m	RcII	Size	AUC
C4.5	28740	9040	18101	3519	0.89	0.67	0.76	0.79	0.82	0.77	0.89	1051.00	0.85
MLP	27805	12973	14168	4454	0.86	0.52	0.68	0.71	0.76	0.67	0.86		0.77
RF	29509	7891	19250	2750	0.91	0.71	0.79	0.82	0.85	0.81	0.91		0.90

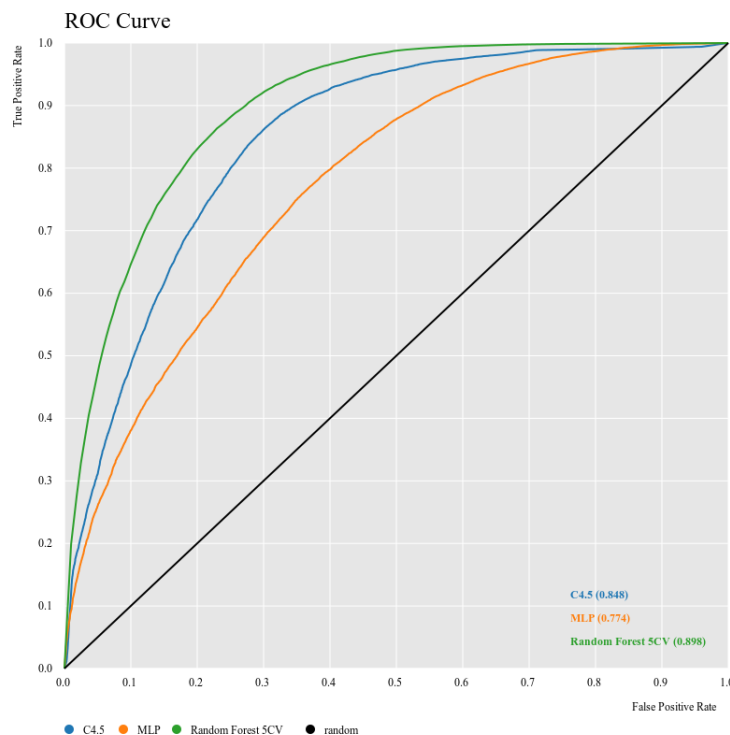


Figura 10: Curva ROC: Clase positiva: funcional

## Clase funcional needs repair

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-m	Rcll	Size	AUC
C4.5	893	641	54442	3424	0.21	0.99	0.58	0.93	0.31	0.45	0.21	1051.00	0.81
MLP	11	12	55071	4306	0.00	1.00	0.48	0.93	0.01	0.05	0.00		0.69
RF	1217	811	54272	3100	0.28	0.99	0.60	0.93	0.38	0.53	0.28		0.85

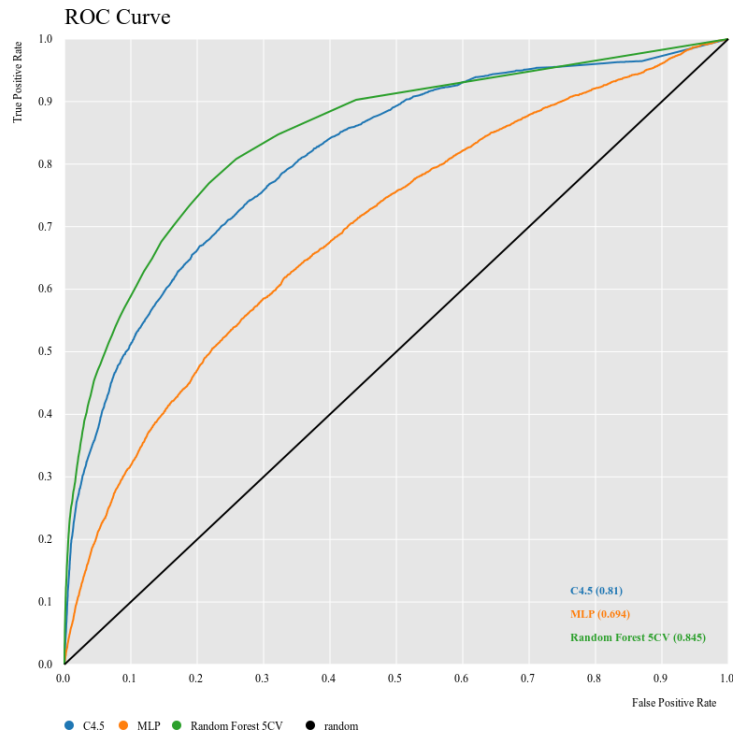


Figura 11: Curva ROC: Clase positiva: funcional needs repair

## Clase non functional

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-m	Rcll	Size	AUC
C4.5	16241	3845	32731	6583	0.71	0.89	0.81	0.82	0.76	0.80	0.71	1051.00	0.87
MLP	13341	5258	31318	9483	0.58	0.86	0.72	0.75	0.64	0.71	0.58		0.80
RF	17155	2817	33759	5669	0.75	0.92	0.86	0.86	0.80	0.83	0.75		0.92

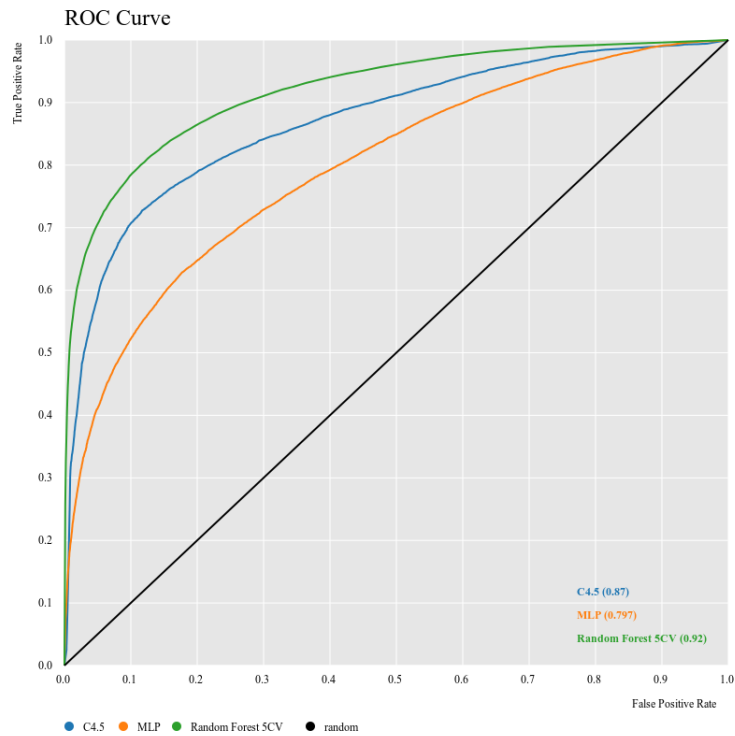


Figura 12: Curva ROC: Clase positiva: non functional

## Análisis

Como podemos ver, la poda en el C4.5 ha eliminado un gran número de hojas (de una media de 6366,4 hojas hemos bajado a una media de 1051 hojas) y además ha mejorado el acierto en todas las clases. En este caso ha resultado que un árbol de decisión más simple ha sido capaz de extraer mejor el conocimiento del dataset.

En cuanto a MLP, no ha mejorado prácticamente nada al añadirle una capa más. Parece que este problema necesita una gran cantidad de capas para ser capaces de extraer el conocimiento de este dataset.

## Procesado de datos

### Pasos seguidos

El primer paso que se ha hecho para preprocesar los datos ha sido eliminar dos columnas:

- 1º **date\_recorded**: Este atributo refleja la fecha en la que se introdujo cada instancia en el dataset. Esta información no aporta ninguna información de la fecha en la que se rompieron las bombas de agua (puede que se rompiese un tiempo antes de que se introdujese en el dataset).
- 2º **data\_recorded\_by**: Este atributo contiene la empresa que registro los datos. En todas las instancias es la misma.

El siguiente paso ha sido estudiar la correlación entre los distintos atributos y eliminar aquellos atributos que tengan una correlación lineal igual a uno (esto quiere decir que las dos columnas son iguales y podemos eliminar una de ellas).

La matriz de correlación es la siguiente:

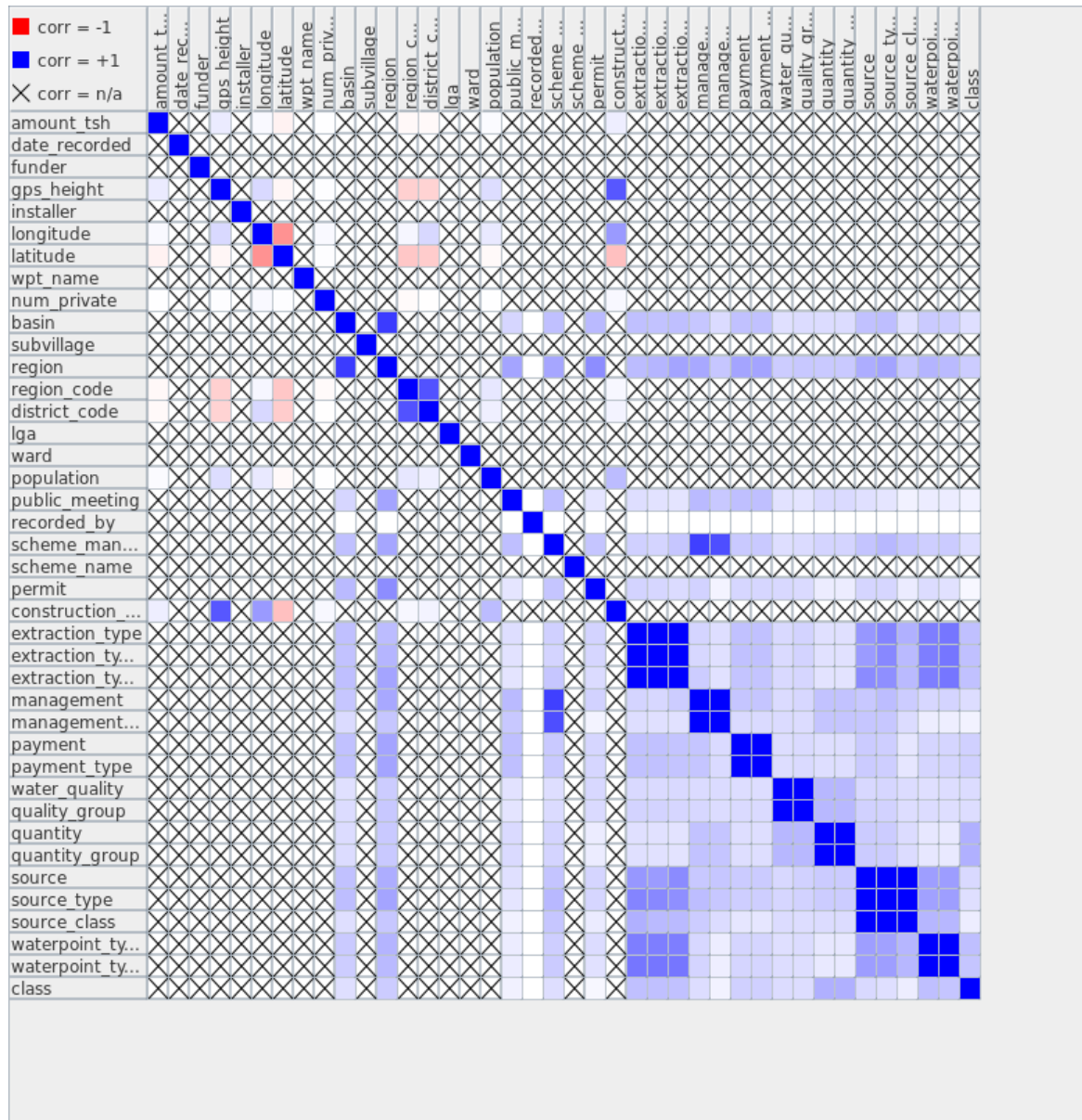


Figura 13: Matriz de correlación lineal

Una vez hemos obtenido esta matriz, con el nodo *correlation filter* podemos usar el modelo de la correlación lineal y eliminar aquellas columnas que tengan una correlación igual a 1. Las filas que se han eliminado han sido: *extraction\_type\_group*, *payment\_type*, *quality\_group*, y *source\_class*.

Debido al desbalance de clase, se ha usado el nodo *equal size sampling* para equilibrar las clases.

Para ver estadísticas del dataset se ha usado el nodo *Data explorer* y después de aplicar el preprocesado el dataset se ha quedado con 12892 instancias y 33 columnas.

Una vez hecho esto se ha vuelto a aplicar los algoritmos con los parámetros por defecto. Los resultados se exponen a continuación.

## Resultados

### Clase functional

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-m	Rcll	Size	AUC
C4.5	2571	1698	6778	1606	0.62	0.80	0.60	0.74	0.61	0.70	0.62	2108.60	0.72
NB	913	613	7996	3370	0.21	0.93	0.60	0.69	0.31	0.44	0.21		0.74
RF	2948	1572	7037	1335	0.69	0.82	0.65	0.77	0.67	0.75	0.69		0.85
MLP	2321	1982	6627	1962	0.54	0.77	0.54	0.69	0.54	0.65	0.54		0.73
KNN	2303	1477	7132	1980	0.54	0.83	0.61	0.73	0.57	0.67	0.54		0.74
XGB	2280	2042	6567	2003	0.53	0.76	0.53	0.69	0.53	0.64	0.53		0.71

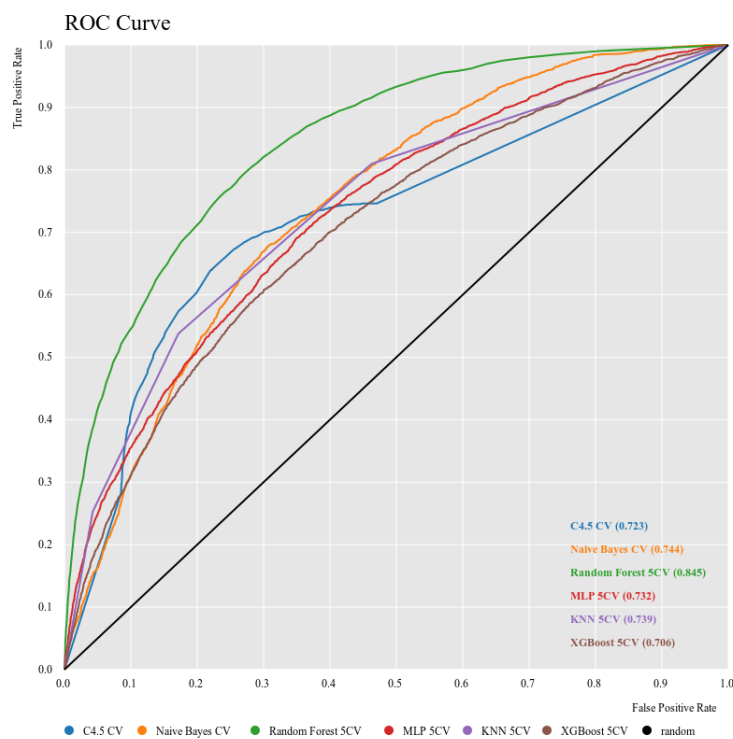


Figura 14: Curva ROC: Clase positiva: functional

### Clase functional needs repair

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-m	Rcll	Size	AUC
C4.5	2886	1506	6882	1379	0.68	0.82	0.66	0.77	0.67	0.75	0.68	2108.60	0.77
NB	3172	3679	4896	1145	0.73	0.57	0.46	0.63	0.57	0.65	0.73		0.73
RF	3209	1359	7216	1108	0.74	0.84	0.70	0.81	0.72	0.79	0.74		0.88
MLP	2582	1965	6610	1735	0.60	0.77	0.57	0.71	0.58	0.68	0.60		0.75
KNN	3095	2360	6215	1222	0.72	0.72	0.57	0.72	0.63	0.72	0.72		0.78
XGB	2433	2034	6541	1884	0.56	0.76	0.54	0.70	0.55	0.66	0.56		0.73

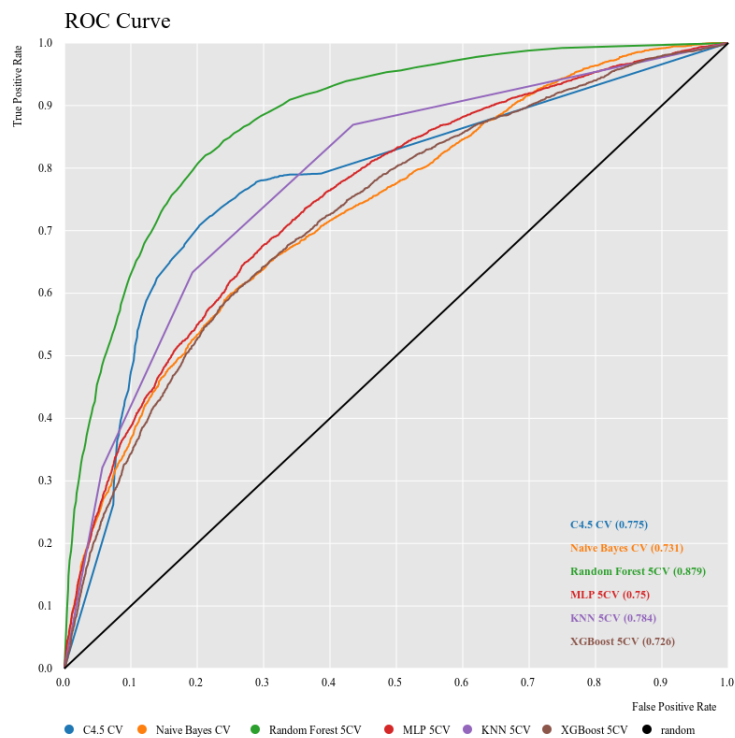


Figura 15: Curva ROC: Clase positiva: functional needs repair

## Clase non functional

ALG	TP	FP	TN	FN	TPR	TNR	PPV	AC	F1	G-m	Rcll	Size	AUC
C4.5	2869	1123	7319	1342	0.68	0.87	0.72	0.81	0.70	0.77	0.68	2108.60	0.80
NB	2687	1828	6772	1605	0.63	0.79	0.60	0.73	0.61	0.70	0.63		0.79
RF	3008	796	7804	1284	0.70	0.91	0.79	0.84	0.74	0.80	0.70		0.89
MLP	2408	1634	6966	1884	0.56	0.81	0.60	0.73	0.58	0.67	0.56		0.76
KNN	2488	1169	7431	1804	0.58	0.86	0.68	0.77	0.63	0.71	0.58		0.79
XGB	2295	1808	6792	1997	0.53	0.79	0.56	0.70	0.55	0.65	0.53		0.73

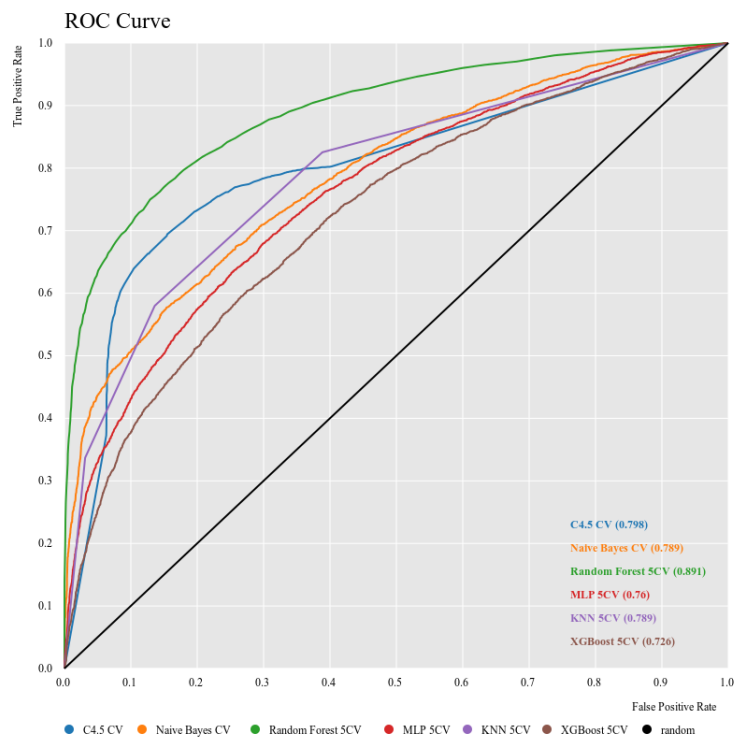


Figura 16: Curva ROC: Clase positiva: non functional

## Sobre-aprendizaje

También se ha estudiado el sobre-aprendizaje en estos algoritmos y los resultados han sido los siguientes:

### ■ C4.5

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.92	0.70	1.31
non functional	0.93	0.77	1.21
functional needs repair	0.93	0.75	1.25

### ■ Naive Bayes

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.45	0.44	1.02
non functional	0.71	0.70	1.01
functional needs repair	0.65	0.65	1.00

### ■ Random Forest

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.96	0.75	1.27
non functional	0.97	0.80	1.21
functional needs repair	0.96	0.79	1.21

**■ Multilayer Perceptron**

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.66	0.65	1.01
non functional	0.70	0.68	1.02
functional needs repair	0.69	0.68	1.01

**■ KNN**

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.82	0.67	1.22
non functional	0.84	0.71	1.19
functional needs repair	0.84	0.72	1.17

**■ XGBoosting**

Class	G-mean training	G-mean test	Overfitting Ratio
functional	0.64	0.64	1.00
non functional	0.65	0.65	1.00
functional needs repair	0.66	0.66	1.00

## Análisis

Como vemos, tras aplicar el preprocesado, hemos bajado un poco el acierto en la clase mayoritaria (functional), hemos mejorado mucho en la clase intermedia (functional needs repair) y en la clase minoritaria se ha quedado prácticamente igual. Parece que hay muy pocos ejemplos de la clase non functional y las técnicas que se han aplicado no han sido suficientes como para mejorar la capacidad de extraer el conocimiento del dataset.

En cuanto al sobre aprendizaje, parece que los algoritmo C4.5 y Random Forest tienen una diferencia significativa entre la media geométrica de training y test, aunque no parece que los algoritmos hayan sufrido mucho sobre-aprendizaje.

En cambio, el algoritmo KNN si que parece que ha sufrido un poco de sobre-aprendizaje en la clase functional, puntuando un 82 % en training y 67 % en test.