Práctica 1

# Análisis predictivo mediante clasificación

Miguel Ángel Fernández Gutiérrez

**Inteligencia de Negocio**
5º Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

mianfg.me

# Índice

# 1. Introducción

En esta primera práctica aplicaremos algoritmos de clasificación a ciertos problemas. Mediante estos algoritmos y métodos de preprocesado pretenderemos obtener un modelo que nos permita predecir las clases a partir de nuevos datos.

Seguiremos el siguiente proceso para cada *dataset*:

- **Estudio del *dataset*.** Se estudiarán los problemas para saber qué atributo intentamos predecir, y qué es lo que significan cada uno de los atributos. También estudiaremos el reparto de las clases, para deducir si es necesario un balanceado en la fase de preprocesado.
- **Preprocesado.** Se realizará un cierto preprocesado sobre los datos. Para comprobar que se obtienen mejores resultados, realizaremos comparativas en ciertos *datasets*.
- **Entrenamiento.** Ejecutaremos los diversos algoritmos sobre los diversos *datasets*. Realizaremos un estudio de cuáles de ellos son más convenientes. Es importante notar que, antes de ejecutar un algoritmo, deberíamos de realizar un estudio de si es conveniente. En el caso de esta práctica, ejecutaremos todos los algoritmos para todos los *datasets*, dado que es la implementación más rápida de realizar en KNIME. Compensaremos esto con el análisis de adecuación antes mencionado.
- **Análisis de resultados.** Estudiaremos qué algoritmos obtienen mejores resultados para cada *dataset*, y el por qué de cada uno de ellos.

Algunas consideraciones generales para todos los *datasets*:

- La experimentación se realiza en todo momento usando validación cruzada de 5 particiones, usando los nodos *X-Partitioner* y *X-Aggregator*.
- Donde sea necesaria, utilizaremos la semilla aleatoria 7.

## 1.1. Heart Failure Prediction

De acuerdo a la página web referenciada en el guion de prácticas,[1] el *dataset* que vamos a explorar está formado por 11 atributos que pueden ser utilizados para predecir una posible enfermedad cardiovascular.

El atributo objetivo por tanto es `HeartDisease`, siendo éste `0` si no estará enfermo y `1` si lo estará. Tenemos de este modo dos clases. Analizaremos a continuación la distribución de clases de entre los datos proporcionados.

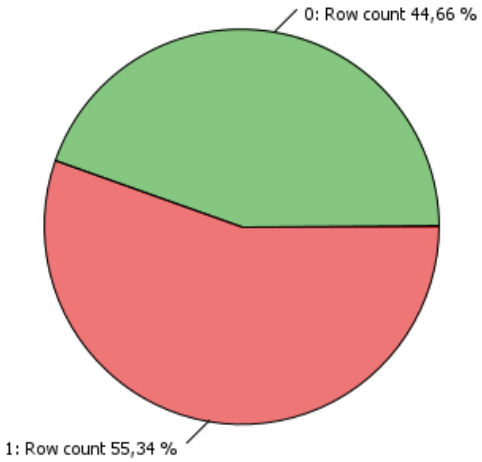


**Figura 1:** Reparto de clases para *Heart Failure Prediction*

Vemos que el reparto está medianamente balanceado.

## 1.2. Mobile Price Classification

Del mismo modo[2] vemos que el *dataset* está formado por 20 atributos, consistentes en características de ciertos móviles, para predecir el rango de precios de entre cuatro categorías (de menor a mayor precio).

Nuestro atributo objetivo es, en este caso, `price_range`. La distribución de clases podemos ver que, para este *dataset*, es perfecta.

[1] Heart Failure Prediction: `https://www.kaggle.com/fedesoriano/heart-failure-prediction`
[2] Mobile Price Classification: `https://www.kaggle.com/iabhishekofficial/mobile-price-classification`

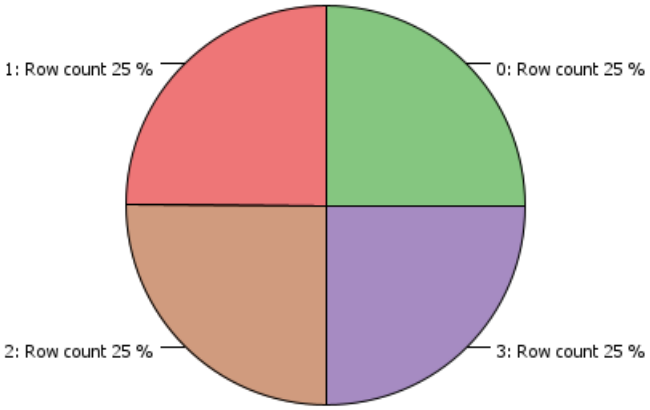


**Figura 2:** Reparto de clases para *Mobile Price Classification*

## 1.3. Bank Marketing

En este *dataset*,[3] tenemos información relacionada con las campañas telefónicas de una institución bancaria portuguesa. El objetivo de esta campaña era vender un producto bancario. Tenemos 20 atributos, y como atributo objetivo tenemos dos clases: `y` si el cliente se ha suscrito al depósito y `no` en caso contrario.

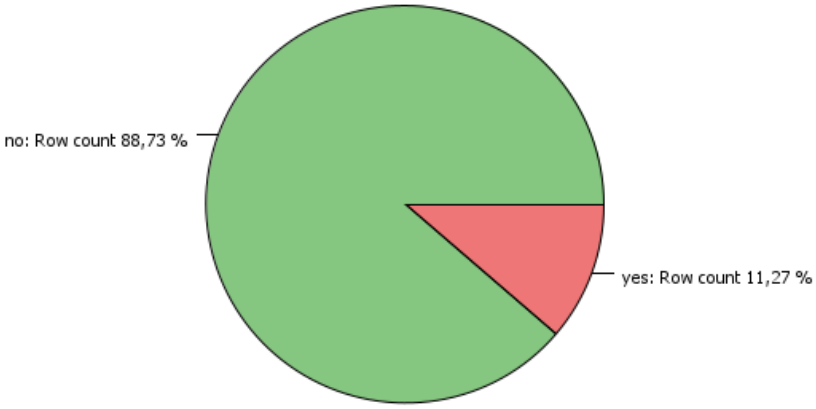


**Figura 3:** Reparto de clases para *Bank Marketing*

[3]Bank Marketing: `https://archive.ics.uci.edu/ml/datasets/bank+marketing`

En este caso, vemos que las clases están claramente desbalanceadas.

### 1.4. Tanzania Water Pump

A continuación[4] estamos interesados en predecir el estado de bombas de agua de Tanzania, para saber si están en estado funcional (`functional`) o no funcional (`non functional`).

Para ello, disponemos de 39 atributos, junto con el atributo que queremos predecir. Vemos que en este caso la distribución de clases no es perfecta, pero al menos mucho mejor que en el caso del *dataset* anterior.

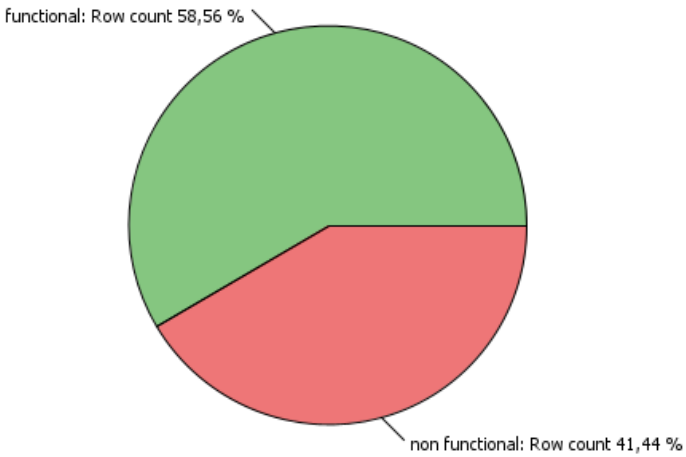


**Figura 4:** Reparto de clases para *Tanzania Water Pump*

## 2. Resultados obtenidos

Para poder realizar adecuadamente las comparaciones, especificaremos en primer lugar cada algoritmo y los resultados obtenidos, para discutir más adelante los errores de todos ellos, así como las curvas ROC, para cada *dataset*, en el apartado *Análisis de resultados*.

El preprocesamiento necesario se explica más adelante, en *Preprocesado de datos*. En este apartado se concreta el preprocesado específico de cada algoritmo (normalización, discretización, etc.), sin entrar en el preprocesado necesario para tratar valores perdidos y atributos redundantes.

[4]Tanzania Water Pump: `https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table`

## 2.1. ZeroR

Comenzamos por el algoritmo **ZeroR**, que no es de gran utilidad práctica pero que nos permite verificar si nuestros algoritmos funcionan correctamente. Este clasificador asigna siempre a los datos la clase mayoritaria. Si a lo largo del desarrollo de esta práctica observamos un algoritmo con peor desempeño que éste, podremos saber que estamos haciendo algo mal. Para implementarlo, creamos el metanodo *ZeroR*.

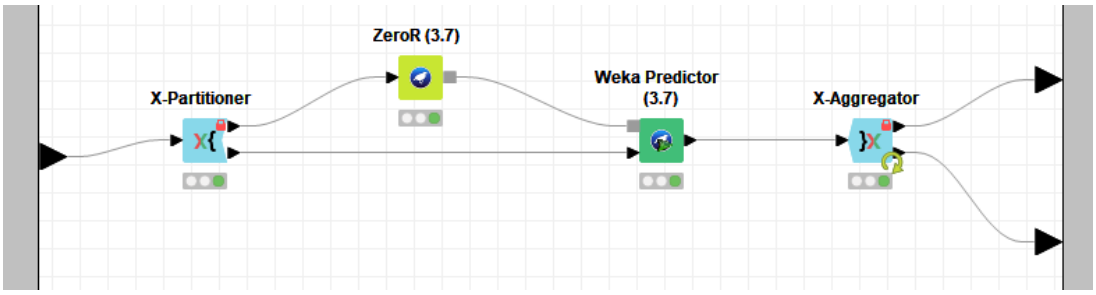


**Figura 5:** Metanodo *ZeroR*

Usando el nodo *Scorer*, vemos la matriz de confusión para cada *dataset*.

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 410 |
| 1 | 0 | 508 |

**(a)** Heart Rate Prediction

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 89 | 114 | 97 | 200 |
| 1 | 98 | 84 | 112 | 205 |
| 2 | 101 | 99 | 91 | 209 |
| 3 | 112 | 103 | 99 | 186 |

**(b)** Mobile Price Classification

| | no | yes |
|---|---|---|
| no | 36548 | 0 |
| yes | 4640 | 0 |

**(c)** Bank Marketing

| | functional | non functional |
|---|---|---|
| functional | 32259 | 0 |
| non functional | 22824 | 0 |

**(d)** Tanzania Water Pump

**Figura 6:** Matrices de confusión para *ZeroR*

## 2.2. Árbol de decisión

Escogemos como primer algoritmo los **árboles de decisión**, que parte de todos los datos y selecciona atributos para dividirlos. Como criterio de división utiliza el índice Gini, ya explicado en clase. Para implementarlo, lo integramos en el metanodo *Árbol de decisión*.

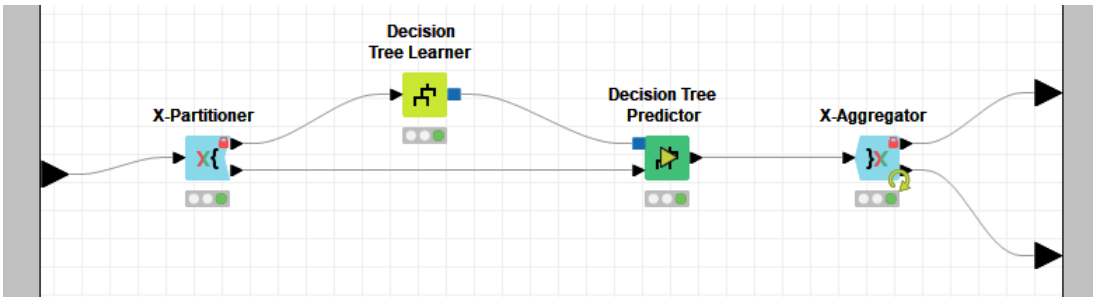


**Figura 7:** Metanodo *Árbol de decisión*

Vemos cómo no hemos realizado ningún procesamiento a los atributos en este nodo. En el caso de las variables continuas, el nodo *Decision Tree Learner* va realizando una agrupación conveniente de estos atributos, para convertirlos en categóricos. Este algoritmo es, además, perfectamente capaz de tratar con valores perdidos.

Veamos a continuación las matrices de confusión.

| | 0 | 1 |
|---|---|---|
| 0 | 329 | 81 |
| 1 | 92 | 416 |

**(a)** Heart Rate Prediction

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 49 | 51 | 0 0 | |
| 1 | 42 | 399 | 59 | 0 |
| 2 | 0 | 52 | 398 | 50 |
| 3 | 0 | 1 | 53 | 446 |

**(b)** Mobile Price Classification

| | no | yes |
|---|---|---|
| no | 34323 | 1979 |
| yes | 2219 | 2243 |

**(c)** Bank Marketing

| | functional | non functional |
|---|---|---|
| functional | 27428 | 4546 |
| non functional | 4894 | 17726 |

**(d)** Tanzania Water Pump

**Figura 8:** Matrices de confusión para *Árbol de decisión*

Es interesante que este árbol nos permitiría crear reglas sencillas que nos ayuden a interpretar los datos fácilmente. Simplemente, debemos ir haciendo una disyunción de cada una de las condiciones que aparecen hasta llegar a un nodo raíz.

## 2.3. k-NN (k-Nearest Neighbors)

Este algoritmo es especialmente interesante para clasificación. El funcionamiento es simple: predice una instancia de acuerdo a sus $k$ vecinos más cercanos. Dado que el cálculo de "cercanía" se hace en función a la distancia euclídea, debemos discretizar y normalizar los datos. Vemos cómo hemos implementado esto haciendo uso del nodo *Category to Number* y *Normalize*.

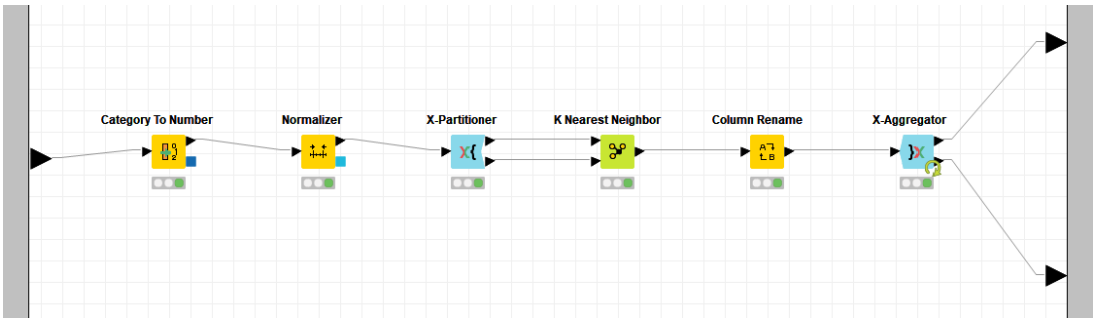


**Figura 9:** Metanodo *k-NN*

Ejecutaremos en primer lugar el algoritmo con $k = 3$, obteniendo de este modo las siguientes matrices de confusión:

| | 0 | 1 |
|---|---|---|
| 0 | 324 | 86 |
| 1 | 66 | 442 |

**(a)** Heart Rate Prediction

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 222 | 155 | 99 | 24 |
| 1 | 130 | 134 | 151 | 85 |
| 2 | 70 | 150 | 148 | 132 |
| 3 | 19 | 94 | 154 | 233 |

**(b)** Mobile Price Classification

| | no | yes |
|---|---|---|
| no | 35260 | 1288 |
| yes | 3211 | 1429 |

**(c)** Bank Marketing

| | functional | non functional |
|---|---|---|
| functional | 27892 | 4367 |
| non functional | 5588 | 17236 |

**(d)** Tanzania Water Pump

**Figura 10:** Matrices de confusión para *Árbol de decisión*

## 2.4. Redes neuronales (NN)

Implementamos a continuación una red neuronal, uno de los algoritmos más conocidos y flexibles pero que, sin embargo, es propicio al *overfitting*. Para ello, usaremos el metanodo *NN*, que se especifica a continuación.

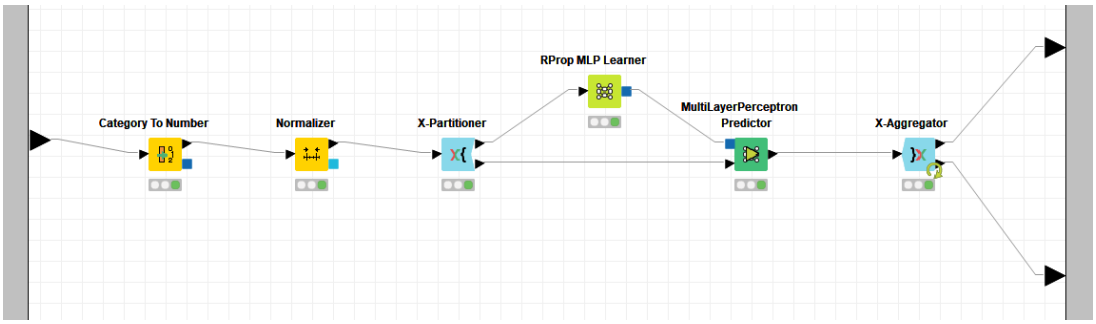


**Figura 11:** Metanodo *NN*

Dado que las redes neuronales realizan combinaciones lineales de los atributos, necesitamos que sean numéricos, y por eso usamos el nodo *Category to Number*. A continuación, realizamos la normalización pertinente con *Normalizer*. Finalmente, ejecutaremos la red neuronal con una capa oculta y 10 neuronas por capa, con un número máximo de 1000 iteraciones. Obtenemos de este modo los siguientes resultados:

| | 0 | 1 |
|---|---|---|
| 0 | 316 | 94 |
| 1 | 65 | 443 |

**(a)** Heart Rate Prediction

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 490 | 10 | 0 | 0 |
| 1 | 11 | 472 | 17 | 0 |
| 2 | 0 | 13 | 473 | 14 |
| 3 | 0 | 1 | 15 | 484 |

**(b)** Mobile Price Classification

| | no | yes |
|---|---|---|
| no | 35244 | 1304 |
| yes | 2221 | 2419 |

**(c)** Bank Marketing

| | functional | non functional |
|---|---|---|
| functional | 28673 | 3586 |
| non functional | 8465 | 14359 |

**(d)** Tanzania Water Pump

**Figura 12:** Matrices de confusión para *Árbol de decisión*

## 2.5. Naïve Bayes

El algoritmo **Naïve Bayes** es otro algoritmo de clasificación, basado en el teorema de Bayes, y asumiendo que todos los atributos son estadísticamente independientes (algo que no es cierto en la mayoría de ocasiones). Calcula la clase más probable, habiendo condicionado al valor del resto de los atributos. Lo implementamos con el nodo *Naive Bayes*.

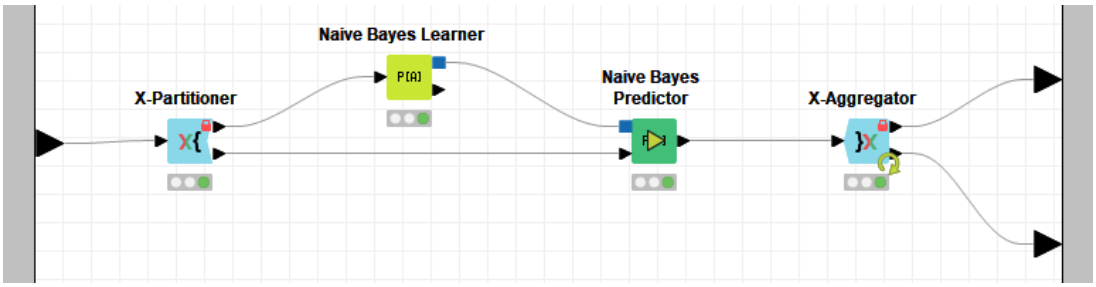


**Figura 13:** Metanodo *Naive Bayes*

Ejecutando este algoritmo obtenemos las siguientes matrices de confusión.

| | 0 | 1 |
|---|---|---|
| 0 | 339 | 71 |
| 1 | 67 | 441 |

**(a)** Heart Rate Prediction

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 432 | 53 | 1 | 14 |
| 1 | 49 | 345 | 97 | 9 |
| 2 | 11 | 91 | 358 | 40 |
| 3 | 8 | 0 | 55 | 437 |

**(b)** Mobile Price Classification

| | no | yes |
|---|---|---|
| no | 31317 | 5231 |
| yes | 1716 | 2924 |

**(c)** Bank Marketing

| | functional | non functional |
|---|---|---|
| functional | 22203 | 10056 |
| non functional | 5594 | 17230 |

**(d)** Tanzania Water Pump

**Figura 14:** Matrices de confusión para *Árbol de decisión*

## 2.6. Random Forest

Como último algoritmo, usaremos un clasificador multiclase, que combina varios clasificadores de una sola clase para mejorar sucesivamente la clasificación, usando *bagging* (cada clasificador actúa de forma independiente). La idea es estabilizar los algoritmos (hacer que con leves cambios en los datos de entrada los resultados de salida no cambien sustancialmente).

**Random Forest** realiza clasificaciones con árboles que no consideran todas las variables ni podan, con subconjuntos de los datos diferentes en cada iteración. Lo implementamos con el metanodo *Random Forest*.

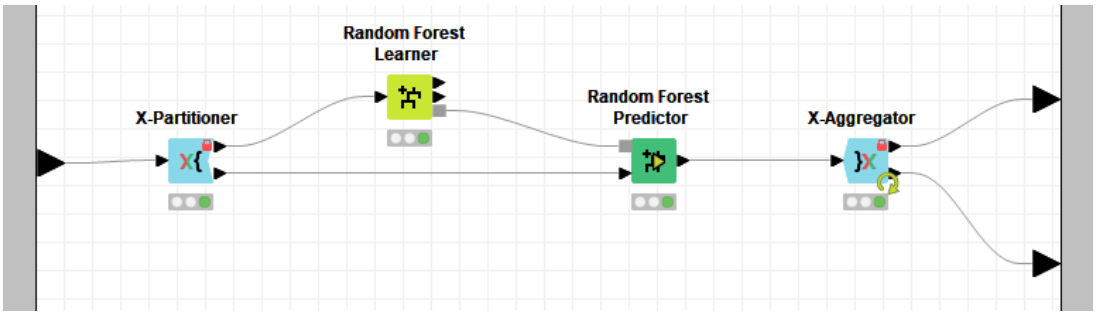


**Figura 15:** Metanodo *Random Forest*

El resultado de ejecutar este algoritmo sobre los cuatro *datasets* puede verse en las matrices de confusión que aparecen a continuación.

| | 0 | 1 |
|---|---|---|
| 0 | 341 | 69 |
| 1 | 52 | 456 |

**(a)** Heart Rate Prediction

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 480 | 20 | 0 | 0 |
| 1 | 36 | 424 | 40 | 0 |
| 2 | 0 | 60 | 399 | 41 |
| 3 | 0 | 0 | 39 | 461 |

**(b)** Mobile Price Classification

| | no | yes |
|---|---|---|
| no | 35527 | 1021 |
| yes | 2559 | 2081 |

**(c)** Bank Marketing

| | functional | non functional |
|---|---|---|
| functional | 29995 | 2264 |
| non functional | 5635 | 17189 |

**(d)** Tanzania Water Pump

**Figura 16:** Matrices de confusión para *Árbol de decisión*

# 3. Análisis de resultados

Es difícil extraer conclusiones al ver el *output* de los algoritmos por separado. Para poder analizar los resultados con más facilidad, analizaremos los errores y curvas ROC para cada *dataset*.

## 3.1. Heart Failure Prediction

| Algorithm | TP | FP | TN | FN | Recall | PPV | TPR | TNR | F1-score | Accuracy | G-mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ZeroR | 0 | 0 | 508 | 410 | 0 | ? | 0 | 1 | ? | 0.553 | 0 |
| Decision Tree | 329 | 92 | 416 | 81 | 0.802 | 0.781 | 0.802 | 0.819 | 0.792 | 0.812 | 0.811 |
| k-NN | 324 | 66 | 442 | 86 | 0.79 | 0.831 | 0.79 | 0.87 | 0.81 | 0.834 | 0.829 |
| NN | 316 | 65 | 443 | 94 | 0.771 | 0.829 | 0.771 | 0.872 | 0.799 | 0.827 | 0.82 |
| Naive Bayes | 339 | 67 | 441 | 71 | 0.827 | 0.835 | 0.827 | 0.868 | 0.831 | 0.85 | 0.847 |
| Random For... | 341 | 52 | 456 | 69 | 0.832 | 0.868 | 0.832 | 0.898 | 0.849 | 0.868 | 0.864 |

**Figura 17:** Errores para *Heart Failure Prediction*

Estudiaremos primero la matriz de confusión. Nuestra idea es maximizar los elementos bien clasificados, es decir, los que tienen un mayor TP (que también tendrán una mayor *accuracy*), así como minimizar los FN.

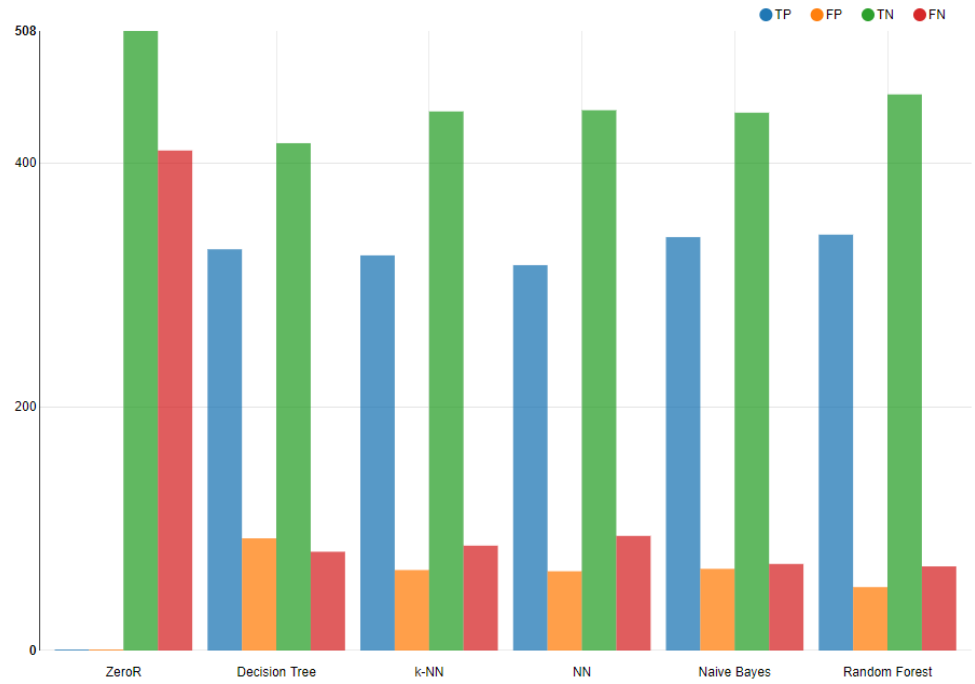


**Figura 18:** Parámetros de la matriz de confusión en *Heart Failure Prediction*

Vemos que los algoritmos que mejor desempeñan son Naïve Bayes y Random Forest, en ese orden. Es interesante ver cómo el algoritmo NN ha sido el que peor ha desempeñado, posiblemente dado que el número de iteraciones impuestas es limitado, así como el número de capas. Aun así, es interesante ver cómo en un *dataset* medianamente sencillo triunfan estas dos opciones. Un motivo posible por el que estos algoritmos desempeñaron mejor es por el uso intrínseco

de variables categóricas. En el caso de NN, debemos pasarlas a números, lo cual propicia un mayor error debido a la naturaleza lineal de este tipo de algoritmos.

Finalmente, nos interesa un algoritmo que aumente los TP a un buen ritmo. Para ello, analizaremos la curva ROC.

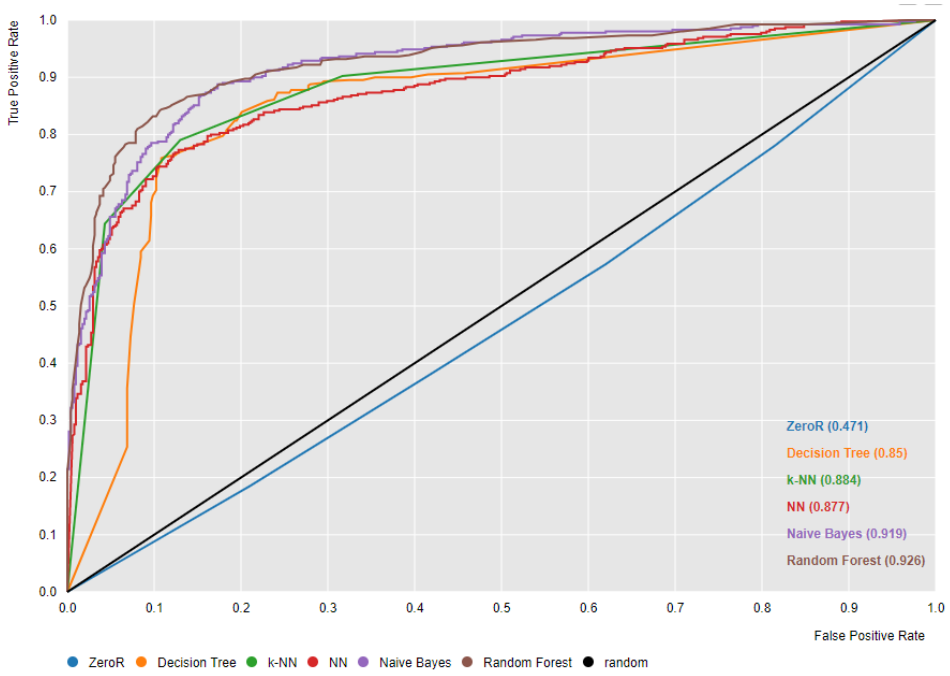


**Figura 19:** ROC para *Heart Failure Prediction*

Los algoritmos con mayor valor AUC (*Area Under the Curve*) son, precisamente, Naïve Bayes y Random Forest, como era de esperar. El peor es, del mismo modo, NN. Además, vemos cómo a NN le cuesta más que al resto de algoritmos incrementar el TP, y que una vez que lo hace se "estanca" mucho más que el resto (le cuesta mucho más crecer).

## Mobile Price Classification

En este caso estamos ante más de dos clases, en concreto cuatro. Veremos primero los errores por clase, y trabajaremos tomando la clase 1 como clase positiva. Dado que la distribución de las clases era homogénea, podremos obtener conclusiones interesantes de este modo.

| Algorithm | Class | TP | FP | TN | FN | Recall | PPV | TPR | TNR | F1-score | Accuracy | G-mean |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ZeroR | 0 | 89 | 311 | 1189 | 411 | 0.178 | 0.223 | 0.178 | 0.793 | 0.198 | 0.639 | 0.376 |
| ZeroR | 1 | 84 | 316 | 1184 | 416 | 0.168 | 0.21 | 0.168 | 0.789 | 0.187 | 0.634 | 0.364 |
| ZeroR | 2 | 91 | 309 | 1191 | 409 | 0.182 | 0.228 | 0.182 | 0.794 | 0.202 | 0.641 | 0.38 |
| ZeroR | 3 | 186 | 614 | 886 | 314 | 0.372 | 0.233 | 0.372 | 0.591 | 0.286 | 0.536 | 0.469 |
| Decision Tree | 0 | 449 | 42 | 1458 | 51 | 0.898 | 0.914 | 0.898 | 0.972 | 0.906 | 0.954 | 0.934 |
| Decision Tree | 1 | 399 | 104 | 1396 | 101 | 0.798 | 0.793 | 0.798 | 0.931 | 0.796 | 0.897 | 0.862 |
| Decision Tree | 2 | 398 | 112 | 1388 | 102 | 0.796 | 0.78 | 0.796 | 0.925 | 0.788 | 0.893 | 0.858 |
| Decision Tree | 3 | 446 | 50 | 1450 | 54 | 0.892 | 0.899 | 0.892 | 0.967 | 0.896 | 0.948 | 0.929 |
| k-NN | 0 | 222 | 219 | 1281 | 278 | 0.444 | 0.503 | 0.444 | 0.854 | 0.472 | 0.751 | 0.616 |
| k-NN | 1 | 134 | 399 | 1101 | 366 | 0.268 | 0.251 | 0.268 | 0.734 | 0.259 | 0.618 | 0.444 |
| k-NN | 2 | 148 | 404 | 1096 | 352 | 0.296 | 0.268 | 0.296 | 0.731 | 0.281 | 0.622 | 0.465 |
| k-NN | 3 | 233 | 241 | 1259 | 267 | 0.466 | 0.492 | 0.466 | 0.839 | 0.478 | 0.746 | 0.625 |
| NN | 0 | 490 | 11 | 1489 | 10 | 0.98 | 0.978 | 0.98 | 0.993 | 0.979 | 0.99 | 0.986 |
| NN | 1 | 472 | 24 | 1476 | 28 | 0.944 | 0.952 | 0.944 | 0.984 | 0.948 | 0.974 | 0.964 |
| NN | 2 | 473 | 32 | 1468 | 27 | 0.946 | 0.937 | 0.946 | 0.979 | 0.941 | 0.971 | 0.962 |
| NN | 3 | 484 | 14 | 1486 | 16 | 0.968 | 0.972 | 0.968 | 0.991 | 0.97 | 0.985 | 0.979 |
| Naive Bayes | 0 | 432 | 68 | 1432 | 68 | 0.864 | 0.864 | 0.864 | 0.955 | 0.864 | 0.932 | 0.908 |
| Naive Bayes | 1 | 345 | 144 | 1356 | 155 | 0.69 | 0.706 | 0.69 | 0.904 | 0.698 | 0.851 | 0.79 |
| Naive Bayes | 2 | 358 | 153 | 1347 | 142 | 0.716 | 0.701 | 0.716 | 0.898 | 0.708 | 0.853 | 0.802 |
| Naive Bayes | 3 | 437 | 63 | 1437 | 63 | 0.874 | 0.874 | 0.874 | 0.958 | 0.874 | 0.937 | 0.915 |
| Random For... | 0 | 480 | 36 | 1464 | 20 | 0.96 | 0.93 | 0.96 | 0.976 | 0.945 | 0.972 | 0.968 |
| Random For... | 1 | 424 | 80 | 1420 | 76 | 0.848 | 0.841 | 0.848 | 0.947 | 0.845 | 0.922 | 0.896 |
| Random For... | 2 | 399 | 79 | 1421 | 101 | 0.798 | 0.835 | 0.798 | 0.947 | 0.816 | 0.91 | 0.869 |
| Random For... | 3 | 461 | 41 | 1459 | 39 | 0.922 | 0.918 | 0.922 | 0.973 | 0.92 | 0.96 | 0.947 |

**Figura 20:** Errores para *Mobile Price Classification*

| Algorithm | TP | FP | TN | FN | Recall | PPV | TPR | TNR | F1-score | Accuracy | G-mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ZeroR | 84 | 316 | 1184 | 416 | 0.168 | 0.21 | 0.168 | 0.789 | 0.187 | 0.634 | 0.364 |
| Decision Tree | 399 | 104 | 1396 | 101 | 0.798 | 0.793 | 0.798 | 0.931 | 0.796 | 0.897 | 0.862 |
| k-NN | 134 | 399 | 1101 | 366 | 0.268 | 0.251 | 0.268 | 0.734 | 0.259 | 0.618 | 0.444 |
| NN | 472 | 24 | 1476 | 28 | 0.944 | 0.952 | 0.944 | 0.984 | 0.948 | 0.974 | 0.964 |
| Naive Bayes | 345 | 144 | 1356 | 155 | 0.69 | 0.706 | 0.69 | 0.904 | 0.698 | 0.851 | 0.79 |
| Random For... | 424 | 80 | 1420 | 76 | 0.848 | 0.841 | 0.848 | 0.947 | 0.845 | 0.922 | 0.896 |

**Figura 21:** Errores para *Mobile Price Classification* suponiendo que la clase positiva es 1

Del mismo modo que en el caso anterior, pretendemos maximizar TP y minimizar FN.

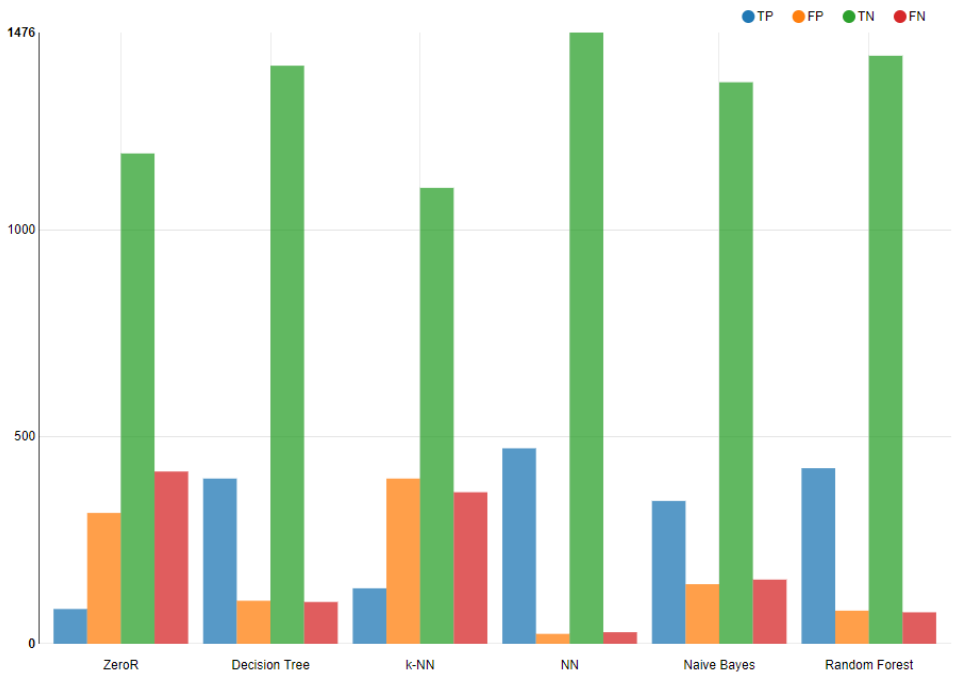


**Figura 22:** Parámetros de la matriz de confusión en *Mobile Price Classification*

En este caso, tenemos que el mejor algoritmo es NN, justo al contrario que en el *dataset* anterior. Además, incurriendo en la comparación con dicho *dataset*, vemos que los algoritmos Naïve Bayes y Random Forest siguen siendo muy buenos, justo por detrás de NN. En este caso, la explicación puede darse a que este *dataset* tiene más atributos numéricos que el anterior, y por tanto permite a NN obtener resultados sustancialmente mejores. Por otra parte, vemos cómo k-NN es el peor algoritmo de los estudiados. Observando la curva ROC podremos obtener más conclusiones.

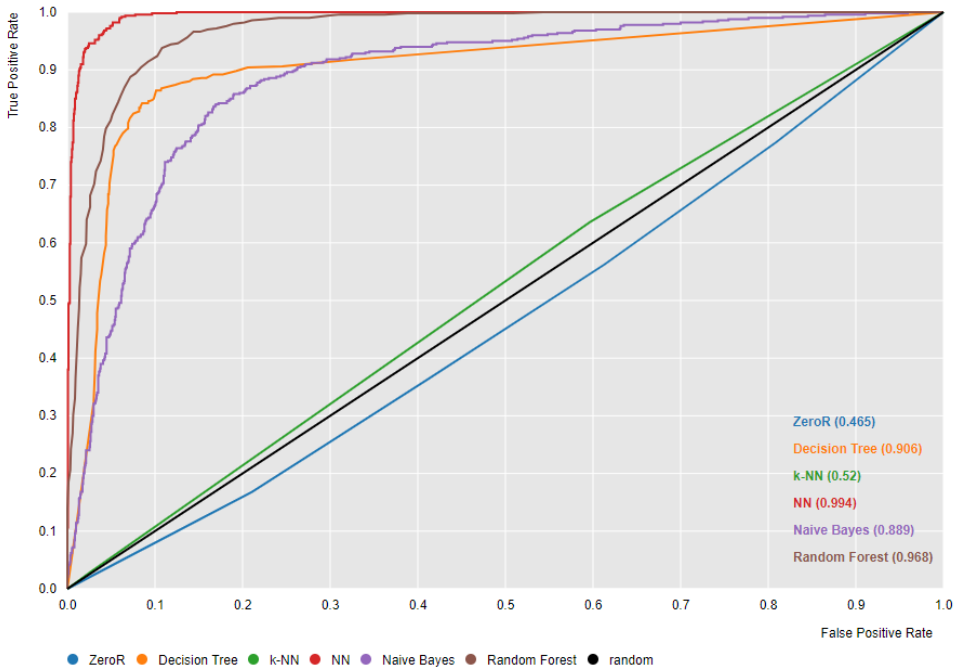


**Figura 23:** ROC para *Mobile Price Classification*

Sin lugar a dudas, NN tiene mayor AUC. Es llamativo ver cómo Árbol de decisión logra incrementar el TP con mayor velocidad que Naïve Bayes, pero regulariza su crecimiento mucho antes, llegando Naïve Bayes a superarlo.

Por otra parte, vemos el mal desempeño de k-NN. Es interesante ver cómo es tan marcado para este *dataset*, y creo que puede deberse a dos motivos. Uno de ellos puede ser la elección de semilla aleatoria. Pero, con más seguridad, puede tratarse al mismo motivo por el que NN desempeñaba mal en el *dataset* anterior. Dado que k-NN mide distancias, nos encontramos ante el problema de medir distancias entre atributos categóricos. Para aplicar k-NN, como hemos dicho anteriormente, hemos debido discretizarlos (pasarlos a números) y normalizarlos. Sin embargo, el orden en el que etiquetemos numéricamente a cada atributo es muy importante para este algoritmo. Cada asignación nos proporcionará diferentes distancias. Es posible que en este caso la asignación haya sido desfavorable, o que el *dataset* tenga los atributos categóricos distribuidos de cierta forma que perjudique gravemente esta técnica.

## Bank Marketing

| Algorithm | TP | FP | TN | FN | Recall | PPV | TPR | TNR | F1-score | Accuracy | G-mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ZeroR | 0 | 0 | 36548 | 4640 | 0 | ? | 0 | 1 | ? | 0.887 | 0 |
| Decision Tree | 2243 | 1979 | 34323 | 2219 | 0.503 | 0.531 | 0.503 | 0.945 | 0.517 | 0.897 | 0.689 |
| k-NN | 1429 | 1288 | 35260 | 3211 | 0.308 | 0.526 | 0.308 | 0.965 | 0.388 | 0.891 | 0.545 |
| NN | 2419 | 1304 | 35244 | 2221 | 0.521 | 0.65 | 0.521 | 0.964 | 0.579 | 0.914 | 0.709 |
| Naive Bayes | 2924 | 5231 | 31317 | 1716 | 0.63 | 0.359 | 0.63 | 0.857 | 0.457 | 0.831 | 0.735 |
| Random For... | 2081 | 1021 | 35527 | 2559 | 0.448 | 0.671 | 0.448 | 0.972 | 0.538 | 0.913 | 0.66 |

**Figura 24:** Errores para *Bank Marketing*

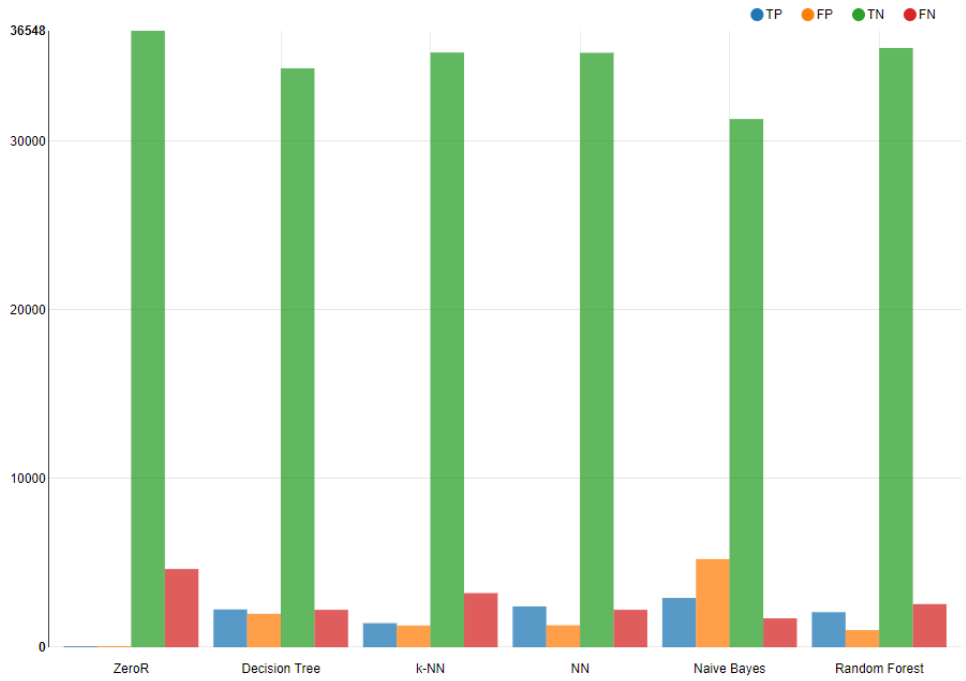


**Figura 25:** Parámetros de la matriz de confusión en *Bank Marketing*

Para este *dataset*, vemos de nuevo que Random Forest es el mejor algoritmo, seguido de NN. En este caso, de nuevo k-NN desempeña una labor pobre, así como el árbol de decisión. Además, el árbol de decisión que obtenemos es de una complejidad considerable, dado el número de atributos.

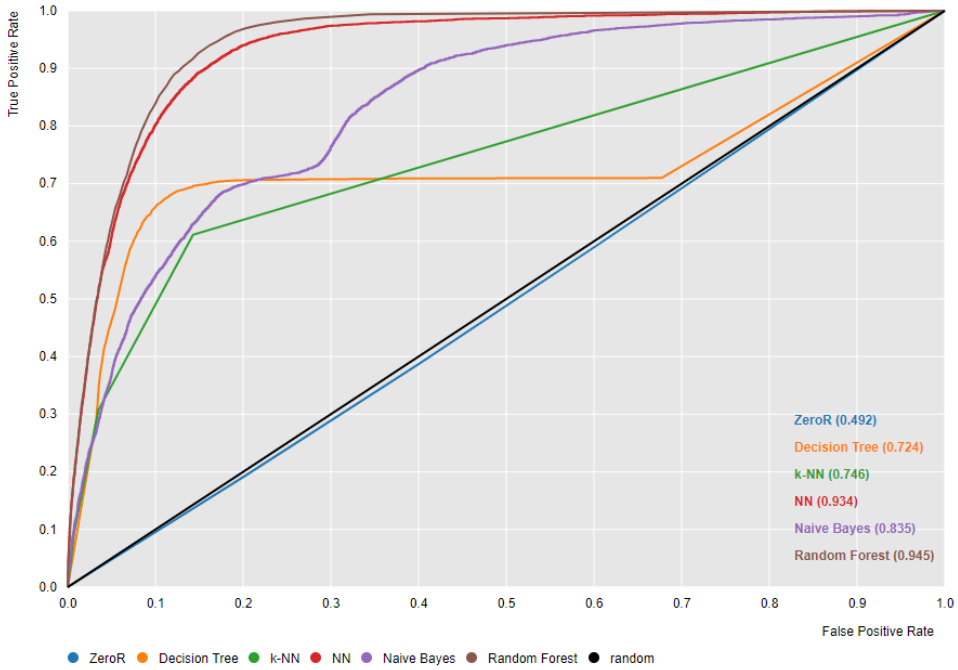


**Figura 26:** ROC para *Bank Marketing*

Mediante el análisis de ROC y AUC podemos obtener conclusiones similares a lo ya descrito. También vemos cómo el crecimiento del árbol de decisión se estabiliza mucho antes de llegar al ratio de TP máximo.

## Tanzania Water Pump

Este *dataset* es, con diferencia, el más exigente. Veamos cómo actúan nuestros algoritmos ante él.

| Algorithm | TP | FP | TN | FN | Recall | PPV | TPR | TNR | F1-score | Accuracy | G-mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ZeroR | 32259 | 22824 | 0 | 0 | 1 | 0.586 | 1 | 0 | 0.739 | 0.586 | 0 |
| Decision Tree | 27428 | 4894 | 17726 | 4546 | 0.858 | 0.849 | 0.858 | 0.784 | 0.853 | 0.827 | 0.82 |
| k-NN | 27892 | 5588 | 17236 | 4367 | 0.865 | 0.833 | 0.865 | 0.755 | 0.849 | 0.819 | 0.808 |
| NN | 28673 | 8465 | 14359 | 3586 | 0.889 | 0.772 | 0.889 | 0.629 | 0.826 | 0.781 | 0.748 |
| Naive Bayes | 22203 | 5594 | 17230 | 10056 | 0.688 | 0.799 | 0.688 | 0.755 | 0.739 | 0.716 | 0.721 |
| Random For... | 29995 | 5635 | 17189 | 2264 | 0.93 | 0.842 | 0.93 | 0.753 | 0.884 | 0.857 | 0.837 |

**Figura 27:** Errores para *Tanzania Water Pump*

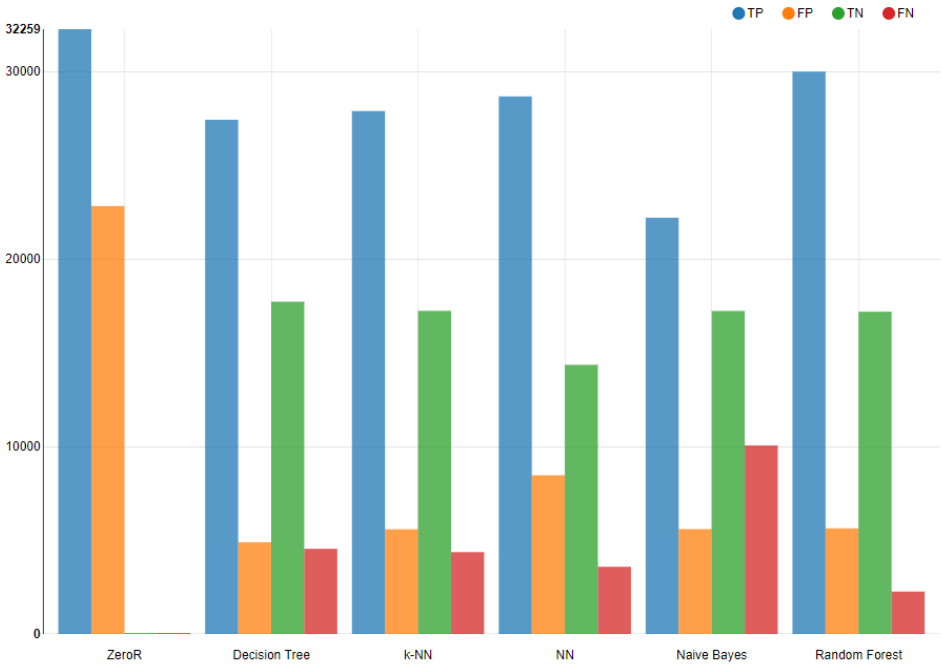


**Figura 28:** Parámetros de la matriz de confusión en *Tanzania Water Pump*

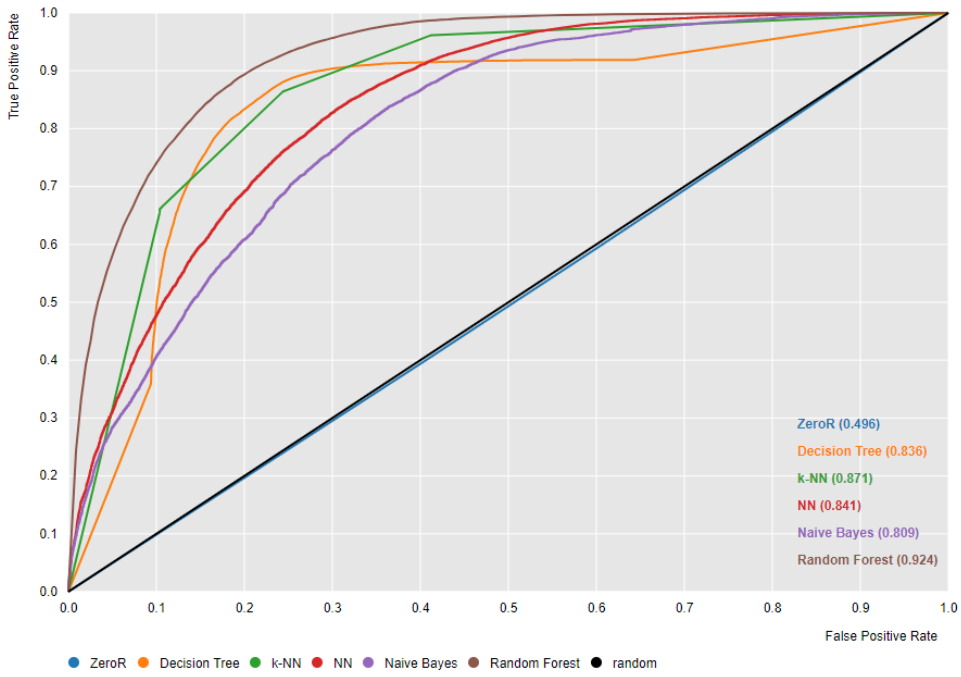


**Figura 29:** ROC para *Tanzania Water Pump*

De nuevo es Random Forest el mejor algoritmo, seguido por NN y, sorprendentemente, por k-NN. La NN ha actuado considerablemente peor en este caso, lo cual puede ser debido al elevado número de atributos y al poco número de iteraciones. El buen comportamiento de k-NN puede deberse en este caso a que disponemos de más atributos numéricos o al comportamiento aleatorio.

Claramente, las medidas de precisión son notablemente inferiores a los *datasets* anteriores. Al encontrarnos ante tantos atributos, es más difícil obtener una regla de clasificación que cubra el mayor número de ejemplos posible.

# 4. Configuración de algoritmos

Estudiaremos seis variaciones de dos algoritmos:

- Variaremos el **número de vecinos** para *nearest neighbors*. Usaremos los valores $k \in \{1, 3, 5\}$, y los llamaremos 1-NN, 3-NN y 5-NN, respectivamente.
- Variaremos el **número de capas ocultas** de la red neuronal, tomando 1, 2 y 3 capas, y los llamaremos NN-1h, NN-2h y NN-3h, respectivamente.

Veamos los resultados obtenidos por todos los algoritmos, para a continuación comentarlos con detalle. En todos ellos usaremos *Mobile Price Classification* como *dataset* para realizar las comparaciones.

| S Algorithm | I TP | I FP | I TN | I FN | D Recall | D PPV | D TPR | D TNR | D F1-score | D Accuracy | D G-mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1-NN | 155 | 372 | 1128 | 345 | 0.31 | 0.294 | 0.31 | 0.752 | 0.302 | 0.641 | 0.483 |
| 3-NN | 134 | 399 | 1101 | 366 | 0.268 | 0.251 | 0.268 | 0.734 | 0.259 | 0.618 | 0.444 |
| 5-NN | 149 | 360 | 1140 | 351 | 0.298 | 0.293 | 0.298 | 0.76 | 0.295 | 0.644 | 0.476 |
| NN-1h | 472 | 24 | 1476 | 28 | 0.944 | 0.952 | 0.944 | 0.984 | 0.948 | 0.974 | 0.964 |
| NN-2h | 471 | 29 | 1471 | 29 | 0.942 | 0.942 | 0.942 | 0.981 | 0.942 | 0.971 | 0.961 |
| NN-3h | 472 | 34 | 1466 | 28 | 0.944 | 0.933 | 0.944 | 0.977 | 0.938 | 0.969 | 0.961 |

**Figura 30:** Errores para configuraciones adicionales

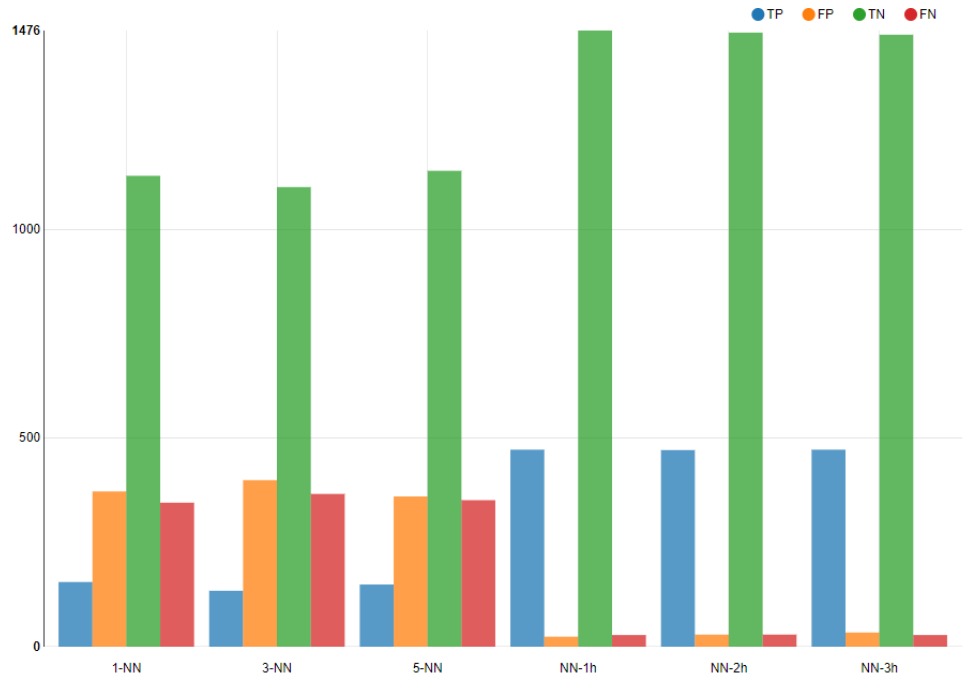


**Figura 31:** Parámetros de la matriz de confusión, configuraciones adicionales

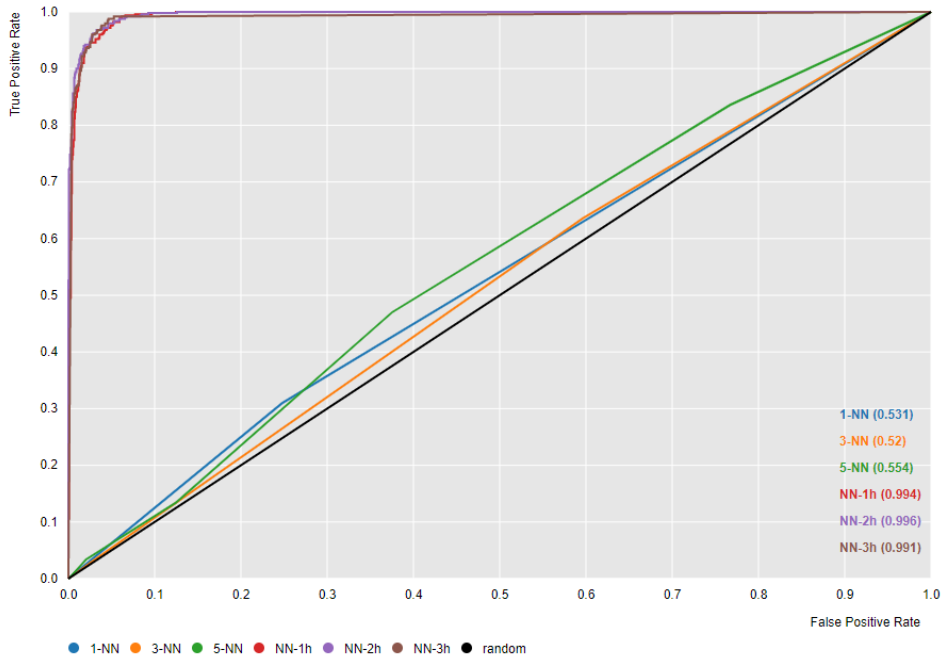


**Figura 32:** ROC para configuraciones adicionales

## 4.1. k-NN

Encontrar el mejor valor de *k* para este algoritmo es, en general, complicado, y depende de cada conjunto de datos. Para nuestros datos, vemos cómo las diferencias de precisión son poco signficativas, siendo 5-NN la variante más precisa.

Vemos que además 1-NN es el que más TP tiene, pero es el que tiene peor rendimiento. Esto se debe a que, al ver un solo vecino, aumentamos la clasificación de positivos, y con ello los FP. Como regla general, vemos que al aumentar el número de vecinos disminuye el TP y también el FP.

De todos modos, vemos que incluso el mejor valor de *k* no supera a los resultados obtenidos por Random Forest o los árboles de decisión.

## 4.2. NN

En este caso vemos cómo incrementar el número de capas ocultas no tiene mucho efecto sobre el resultado. De hecho, vemos cómo NN-1h es el que obtiene los mejores resultados. Esto nos vuelve a indicar cómo encontrar la mejor configuración es un proceso sumamente complicado.

Además, hemos de tener en cuenta la tendencia que tienen las redes neuronales al *overfitting*: aumentar el número de neuronas puede disminuir el error de *train*, pero puede conllevar una menor generalización.

# 5. Procesado de datos

## Heart Failure Prediction

En el caso de *Heart Failure Prediction*, no he visto conveniente realizar ningún preprocesado, dado que el número de atributos es razonable, todos ellos tienen un sentido claro y están bien descritos, y no existen valores perdidos en los datos.

## Mobile Price Classification

Del mismo modo que en el *dataset* anterior, no he considerado necesario realizar ningún preprocesado para este conjunto de datos, por las mismas razones que en *Heart Failure Prediction*. Además, en este caso las clases están perfectamente balanceadas, lo que nos indica que el preprocesado puede haber sido realizado *a priori* y que tenemos datos ya tratados.

## Bank Marketing

Antes de todo observemos que, en la descripción del *dataset*, tenemos ciertos consejos que vamos a seguir a la hora del preprocesado.

En primer lugar, vemos que el *dataset* tiene **valores perdidos** codificados. En concreto, hay muchos atributos (como `job`, `marital` o `loan`) en donde aparece el valor `unknown`, es decir, "desconocido". Trataremos este valor como un valor perdido, de modo que no usemos `unknown` como una clase más del atributo.

A continuación, vemos que en la descripción se nos recomienda **eliminar el atributo** `duration`: "this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model". Seguiremos este consejo en nuestro preprocesado.

Finalmente, trabajaremos con los valores perdidos que especificamos al principio. Para ello, usaremos el siguiente flujo:

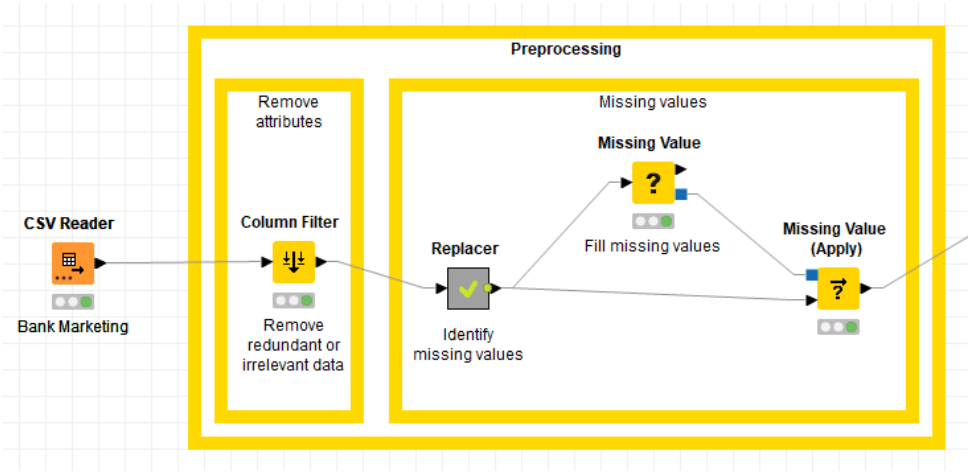


**Figura 33:** Flujo de preprocesamiento para *Bank Marketing*

En *Replacer* usamos sencillamente un nodo *String Manipulation (Column)* con la expresión `toNull(regexReplace($$CURRENTCOLUMN$$, "unknown", ""))`, que nos pasará todos los valores `unknown` a valores perdidos (en KNIME aparecen en las tablas con el símbolo ?). En el nodo *Missing Value* seguimos la estrategia "Most Frequent Value" para los atributos de tipo cadena de caracteres (el resto no es relevante, dado que no tienen valores perdidos).

## Tanzania Water Pump

En este *dataset* tenemos datos mucho más desorganizados. Comenzaremos **eliminando atributos** que son redundantes o no relevantes:

- `funder`: es el nombre de la persona que fundó el pozo de agua, que no tiene relevancia. Además, en algunas ocasiones se escribe el nombre de la misma persona de formas diferentes.
- `wpt_name`: el nombre del pozo de agua no es relevante para nuestro estudio. Dejar este atributo sólo aumentaría la complejidad y el ruido en nuestros modelos.
- `num_private`: lo eliminamos porque no aparece en la descripción del *dataset*, además de que prácticamente todos los valores sean `0`.
- `subvillage`, `region`, lga y `ward`: son redundantes con otros atributos numéricos que nos proporcionan la misma información respecto a la localización geográfica.
- `recorded_by`: es la empresa que realiza el registro de los datos cada vez, lo he eliminado dado que considero que no es relevante pero, además, por problemas similares al caso del atributo `funder`.
- `scheme_name`: eliminado porque es redundante con `scheme_management`.
- `extraction_type_group`, `extraction_type_class`: son redundantes con `extraction_type`, que es el atributo mucho más específico (los otros dos son categorizaciones de éste).
- `quality_group`, `quantity_group`, `waterpoint_type_group`: del mismo modo que en los atributos anteriores, nos proporcionan información ya existente.

Eliminaremos estas columnas del mismo modo que hicimos en el flujo especificado en *Bank Marketing*, mediante el nodo *Column Filter*. A continuación, trataremos los **valores perdidos**. Para ello, basta ver que hay lugares donde no aparecen los datos o aparece un `0` en donde el atributo es de tipo *string*. Volvemos a usar los nodos *String Filter* en el metanodo *Replacer* para tratar con estos valores.

Rellenamos los valores perdidos usando los siguientes criterios: "Most Frequent Value" para *string* y "Mean" para *doubles* e *integers*.

# 6. Interpretación de resultados

Intentaremos interpretar los resultados obtenidos y obtener ciertas conclusiones sobre el desempeño de los algoritmos que hemos probado.

En primer lugar, vemos cómo los algoritmos basados en árboles han logrado mejores resultados que el resto. En efecto, este tipo de algoritmos son especialmente buenos en problemas de clasificación pero, aún más, en problemas donde hay muchas variables categóricas, como era nuestro caso.

Las redes neuronales, en general mucho más potentes, deberían ser usadas preferentemente con atributos continuos, y son más indicadas para tareas de regresión. Por otra parte, k-NN ha sido el que continuamente ha desempeñado peor. Hemos de recordar que este algoritmo es más indicado para *unsupervised learning*, en concreto para problemas de *clustering*.

Finalmente, una observación respecto a árboles y, en especial, su complejidad. Los algoritmos basados en árboles son muy interesantes porque son muy **interpretables**: podemos obtener una serie de reglas de su salida, en contraposición con las redes neuronales, por ejemplo, que se consideran ***cajas negras*** (aunque son interesantes para ver qué atributos tienen más importancia en el *dataset*, al igual que otros algoritmos del estilo). Sin embargo, uno de los principales problemas de éstos es que su complejidad (número de nodos hoja) aumenta mucho conforme tenemos muchos datos y, en especial, cuando tenemos muchos atributos.

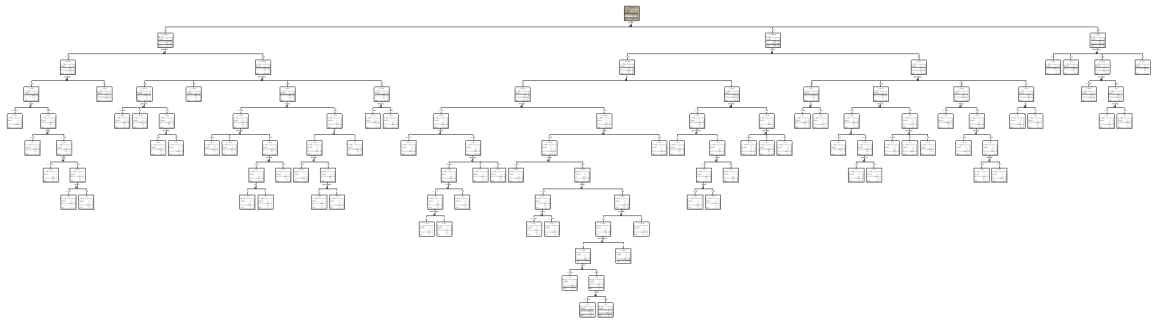

**Figura 34:** Árbol de decisión para *Heart Failure Prediction*

# 7. Bibliografía

KNIME, *From Modeling to Scoring: Confusion Matrix and Class Statistics* . https://www.knime.com/blog/from-modeling-to-scoring-confusion-matrix-and-class-statistics

KNIME, *Simple Preprocessing Example*. https://hub.knime.com/knime/spaces/Examples/latest/02_ETL_Data_Manipulation/00_Basic_Examples/01_Example_for_Standard_Preprocessing~giR07n4V

Jorge Casillas, *KNIME*. https://ccia.ugr.es/~casillas/knime.html