UNIVERSIDAD
DE GRANADA

Inteligencia del Negocio
Grado en Ingeniería Informática
Curso 2018-2019
https://consigna.ugr.es/f/6Ri6jjdgQ9BA5XGz/P1-Ramirez-Garcia-Felix.zip

Memoria Práctica 1 : Grupo 1
Análisis Predictivo Empresarial Mediante Clasificación.

Félix Ramírez García
felixramirezgarcia@correo.ugr.es

4 de noviembre de 2018

Índice

Índice de figuras

Índice de tablas

1. Introducción.

Esta práctica ha sido llevada a cabo para la asignatura Inteligencia del Negocio de la universidad de Granada, asignatura de cuarto curso del Grado en Ingeniería Informática. En ella veremos el uso de algoritmos de aprendizaje supervisado de clasificación para realizar el análisis predictivo de los datos que se describen a continuación.

El contenido de noticias en la red esta creciendo continuamente . Esto hace necesario para cualquier medio de comunicación la predicción del impacto que tengan las noticias en la sociedad . Pudiendo analizar los artículos antes de su publicación se pueden optimizar sus características para que tengan una mayor popularidad.

El conjunto de datos [3] se basa en 39.644 noticias con valores perdidos extraídas en 2015. Cada noticia tiene las siguientes características:

1. n_tokens_title:	Número de palabras en el título
2. n_tokens_content:	Número de palabras en el contenido
3. n_unique_tokens:	Tasa de palabras únicas en el contenido
4. n_non_stop_words:	Velocidad de palabras sin parar en el contenido
5. n_non_stop_unique_tokens:	Ritmo de palabras únicas sin parar en el contento
6. num_hrefs:	Número de enlaces
7. num_self_hrefs:	Número de enlaces a otros artículos publicados
8. num_imgs:	Número de imágenes
9. num_videos:	Número de vídeos
10. longitud_media_del_token:	Longitud media de las palabras en el contento
11. num_keywords:	Número de palabras clave en los metadatos
12. data_channel_is_lifestyle:	¿Es el canal de datos .Estilo de vida¿
13. data_channel_is_entertainment:	¿Es el canal de datos 'Entretenimiento'?
14. data_channel_is_bus:	¿Es el canal de datos'Business'?
15. data_channel_is_socmed:	¿Es el canal de datos 'Social Media'?
16. data_channel_is_tech:	¿Es el canal de datos 'Tech'?
17. data_channel_is_world:	¿Es el canal de datos 'Mundo'?
18. kw_min_min_min:	Palabra clave peor Min
19. kw_max_min:	Palabra clave peor Max
20. kw_avg_min:	Palabra clave peor, promedio
21. kw_min_max:	Mejor palabra clave (min. acciones)
22. kw_max_max:	Mejor palabra clave (porcentaje máximo)
23. kw_avg_max:	Mejor palabra clave (promedio de acciones)
24. kw_min_avg:	Palabra clave media (min. acciones)
25. kw_max_avg:	Palabra clave media (máx. acciones)
26. kw_avg_avg_avg:	Palabra clave media (acciones medias)
27. auto-referencia_compartimentos_mínimos:	Participación mín de los referenciados
28. auto-referencia_compartimentos_max:	Porcentaje máx de referenciados

29. self_reference_avg_sharess: Promedio de acciones de artículos referenciados
30. día-de-la-semana-es-lunes: ¿El artículo fue publicado un lunes?
31. día-de-la-semana-es-el-día-de-la-semana: ¿El artículo fue publicado un martes?
32. día_de_la_semana_es_miércoles: ¿El artículo fue publicado un miércoles?
33. dia-de-la-semana-es-jueves: ¿El articulo se publico un jueves?
34. día-de-la-semana-es-viernes: ¿El artículo fue publicado un viernes?
35. día-de-la-semana-es-sábado: ¿El artículo se publicó un sábado?
36. día-de-la-semana-es-domingo: ¿El artículo se publicó un domingo?
37. is_fin de semana: ¿El artículo fue publicado el fin de semana?
38. LDA_00: Cercanía al tema LDA 0
39. LDA_01: Cercanía al tema 1 de LDA
40. LDA_02: Cercanía al tema 2 de LDA
41. LDA_03: Cercanía al tema 3 de LDA
42. LDA_04: Cercanía al tema 4 de LDA
43. global_subjetividad: Subjetividad del texto
44. global_sentiment_polarity: Polaridad del sentimiento del texto
45. global_rate_positive_words: Índice de palabras positivas en el contenido
46. global_rate_negative_words: Tasa de palabras negativas en el contenido
47. rate_positive_words: Tasa de palabras positivas entre los no neutrales
48. rate_negative_words: Tasa de palabras negativas entre los no neutrales
49. avg_positive_polarity: Polaridad media de las palabras positivas
50. min_polaridad_positiva: Polaridad mínima de las palabras positivas
51. max_polaridad_positiva: Polaridad máxima de las palabras positivas
52. avg_negativo_polaridad: Polaridad media de las palabras negativas
53. min_polaridad_negativa: Polaridad mínima de las palabras negativas
54. max_polaridad_negativa: Polaridad máxima de las palabras negativas
55. title_subjectivity: Subjetividad del título
56. title_sentiment_polarity: Polaridad del título
57. abs_title_subjetividad: Nivel de subjetividad absoluta
58. abs_title_sentiment_polarity: Nivel de polaridad absoluta
59. clase: popular o no_popular (target)

Por lo que se conoce el numero de veces que la noticia ha sido compartida, se puede valorar si la noticia es popular o no. En nuestro caso, fijaremos como umbral 3000 veces, de modo que si la noticia se comparte en un número superior a ese umbral, la noticia es considerada popular.

Principalmente en esta práctica debemos abordar el estudio de los distintos algoritmos de clasificación mediante un diseño experimental apropiado extrayendo conclusiones finales. Para ello debemos aplicar seis algoritmos de clasificación distintos, y a su vez, sobre algunos de ellos, realizar las modificaciones que se consideren oportunas para estudiar los comportamientos de los mismos sobre el problema que se estudia.

Toda la experimentación se ha realizado sobre validación cruzada de 5 particiones. También se han extraído tablas comparativas para mostrar las diferencias de los algoritmos, gráficas ROC , matrices de confusión , G-mean , etc ...

2. Resultados obtenidos.

Esta sección incluye un un apartado por cada uno de los seis algoritmos utilizados. Para una primera toma de contacto , estos algoritmos han sido ejecutados con las opciones por defecto. Así como se han filtrado algunas columnas y se han rellenado valores perdidos con parámetros que han sido seleccionados con aleatoriedad (Como la media) ,sin ningún estudio previo de los datos.

El flujo de trabajo de todos los algoritmos comienza en el nodo lector del fichero , donde se cargan los datos. La imagen 2.1 muestra el Workflow utilizado en KNIME para la ejecución y visualización del resultado de los algoritmos que detallamos en las siguientes subsecciones :

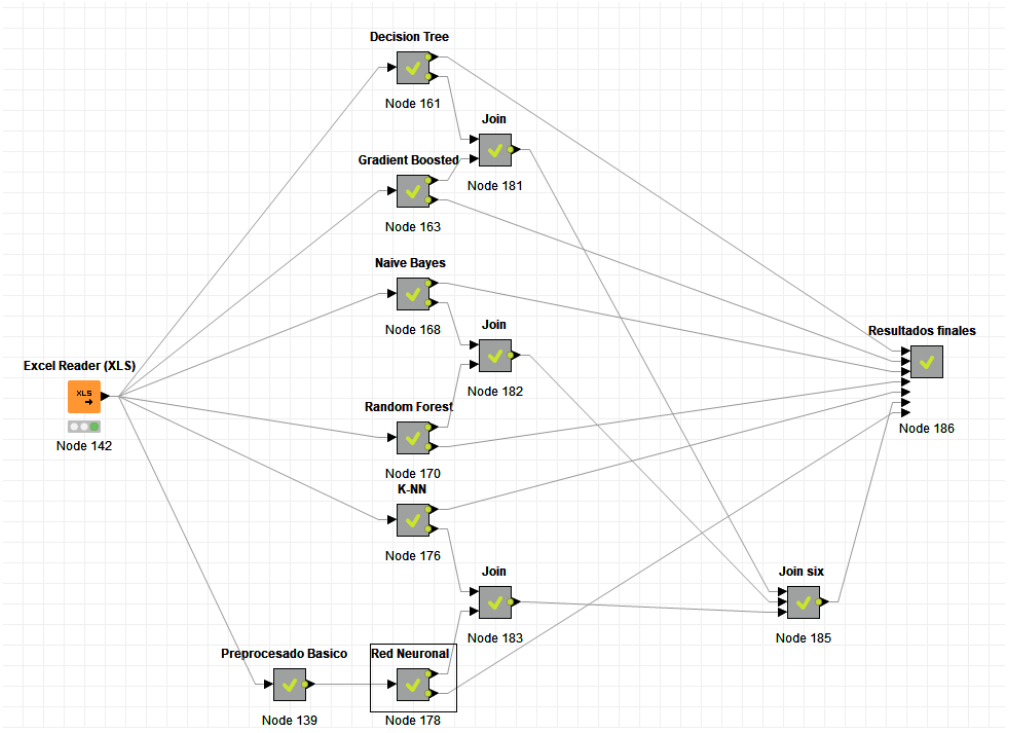

Figura 2.1: Workflow en KNIME

2.1. Decision Tree

En primer lugar usaremos el algoritmo de clasificación Decision Tree (Árbol de decisión) [4] , el cual a partir de un conjunto de datos permite determinar a que clase pertenece el caso de estudio. La figura 2.2 muestra el Workflow utilizado en KNIME :

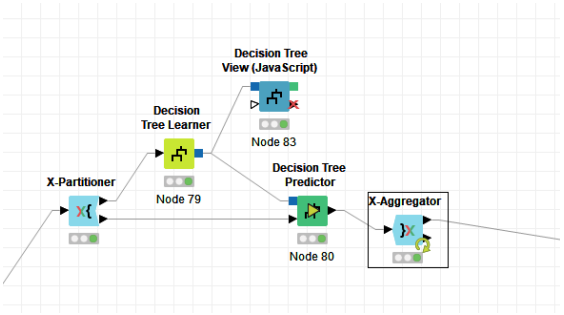

Figura 2.2: Workflow del algoritmo Decision Tree

Dentro de los nodos usados para el algoritmo tenemos X-partitioner , que básicamente nos permite configurar el numero de validaciones (5 en nuestro caso) y el muestreo estratificado. Después tenemos los nodos del algoritmo , en primer lugar el Decision Tree Learner (nodo encargado del entrenamiento) , el cuál se encarga de recoger los datos de entrenamiento e inducir a partir de ellos un árbol de decisión, y en segundo lugar tenemos el nodo Decision Tree Predictor , el cual una vez obtenido un árbol de decisión se lo pasamos por el puerto de color azul, y por el otro puerto le pasamos el conjunto de datos de test , para así predecir el valor de la clase de nuevas muestras. Luego tenemos el nodo X-agregator , que se encarga de volver a reproducir los nodos anteriores como si un bucle se tratara, y también se encarga de sacar una tabla con las particiones y la tabla de errores. Por ultimo usamos el nodo Scorer , que nos permite mostrar la matriz de confusión y las estadísticas del algoritmo.

Para la obtención de los resultados de cada algoritmo se ha realizado en KNIME el Workflow que se muestra en la figura 2.3 ,y ya que se ha realizado el mismo flujo de trabajo para todos los algoritmos solo se va a exponer en esta subsección. En primer lugar nos encontramos con el nodo Scorer , que nos permite mostrar la matriz de confusión y las estadísticas del algoritmo. Los nodos posteriores tienen la función de filtrar estos datos y cambiar el identificador de la clase a predecir por el nombre del algoritmo que se este utilizando, de tal forma que podamos identificar los resultados obtenidos por cualquier algoritmo.

A continuación se muestran el siguiente contenido obtenido por Decision Tree :
Una tabla con la matriz de confusión. (Tabla 2.1)
Una imagen con la tabla de errores .(Figura 2.4)
Una imagen para la curva ROC. (Figura 2.5)
Una tabla con los datos resultantes de la ejecución del algoritmo (Tabla 2.2)

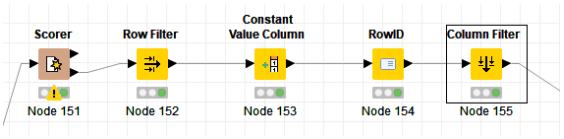

Figura 2.3: Workflow para obtener los resultados de cada algoritmo

x	Popular	NoPopular
Popular	2465	6456
Nopopular	5261	25450

Tabla 2.1: Matriz de confisión de Decision Tree

Row ID	Error in %	Size of Test Set	Error Count
fold 0	30.294	7929	2402
fold 1	29.865	7929	2368
fold 2	28.957	7929	2296
fold 3	29.272	7929	2321
fold 4	29.541	7928	2342

Figura 2.4: Errores de Decision Tree

2.2. Gradient Boosted

El segundo algoritmo que aplicamos es Gradient Boosted [6] , el cual utiliza árboles de regresión para construir un conjunto de árboles. El Workflow del algoritmo Gradient Boosted usado en KNIME es el que muestra la figura 2.6 .

Dentro de los nodos usados para el algoritmo tenemos X-partitioner , que básicamente nos permite configurar el numero de validaciones (5 en nuestro caso) y el muestreo estratificado. Después tenemos los nodos del algoritmo , en primer lugar el Gradient Boosted Trees Learner (nodo encargado del entrenamiento) , y en segundo lugar tenemos el nodo Decision Gradient Boosted Trees Predictor , el cual clasifica los datos usando un modelo de Gradient Boosted Trees , para así predecir el valor de la clase de nuevas muestras. Luego tenemos el nodo X-agregator , que se encarga de volver a reproducir los nodos anteriores como si un bucle se tratara, y también se encarga de sacar una tabla con las particiones y la tabla de errores. Por ultimo usamos el nodo Scorer , que nos permite mostrar la matriz de confusión y las estadísticas del algoritmo.

A continuación se muestra el siguiente contenido obtenido por Gradient Boosted Decision Tree :

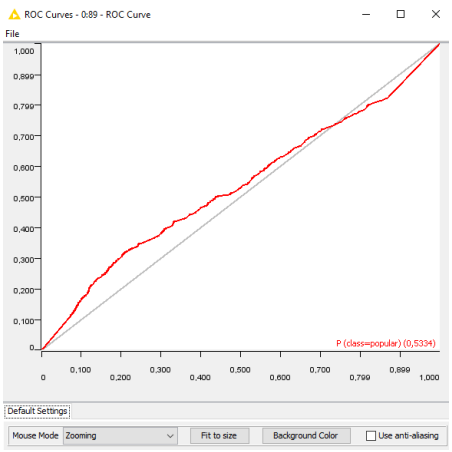

Figura 2.5: Curva ROC de Decision Tree

Medida	Valor
Accuracy	0,704
TPR	0,276
FPR	0,171
TNR	0,829
FNR	0,723
AUC	0,548
F1-Score	0,296
G-Mean	0,479

Tabla 2.2: Resultados de Decision Tree

Una tabla con la matriz de confusión. (Tabla 2.3)
Una imagen con la tabla de errores. (Figura 2.7)
Una imagen para la curva ROC. (Figura 2.8)
Una tabla con los datos resultantes de la ejecución del algoritmo (Tabla 2.4)

2.3. Random Forest

El tercer algoritmo que usaremos será Random Forest [8] , el cuál es una combinación de arboles predictivos tal que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos.
El Workflow del algoritmo Random Forest usado en KNIME es el que muestra la figura 2.9 .
Dentro de los nodos usados para el algoritmo tenemos X-partitioner , que básicamente nos permite configurar el numero de validaciones (5 en nuestro caso) y el muestreo es-

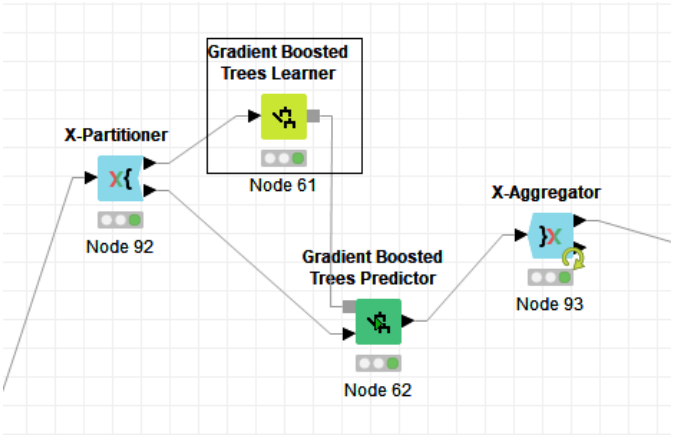

Figura 2.6: Workflow del algoritmo Gradient Boosted

x	Popular	NoPopular
Popular	865	8061
Nopopular	697	30021

Tabla 2.3: Matriz de confisión de Gradient Boosted Decision Tree

tratificado. Después tenemos los nodos del algoritmo , en primer lugar el Random Forest Learner (nodo encargado del entrenamiento) , y en segundo lugar tenemos el nodo Random Forest Predictor , el cual predice patrones de acuerdo a los árboles individuales en un modelo de bosque aleatorio. Luego tenemos el nodo X-agregator , que se encarga de volver a reproducir los nodos anteriores como si un bucle se tratara, y también se encarga de sacar una tabla con las particiones y la tabla de errores. Por ultimo usamos el nodo Scorer , que nos permite mostrar la matriz de confusión y las estadísticas del algoritmo.

A continuación se muestra el siguiente contenido obtenido por Random Forest :
Una imagen con la matriz de confusión. (Figura 2.10)
Una imagen con la tabla de errores. (Figura 2.11)
Una imagen para la curva ROC. (Figura 2.12)
Una tabla con los datos resultantes de la ejecución del algoritmo (Tabla 2.5)

Row ID	D Error in %	I Size of Test Set	I Error Count
fold 0	22.02	7929	1746
fold 1	22.121	7929	1754
fold 2	22.348	7929	1772
fold 3	21.869	7929	1734
fold 4	22.099	7928	1752

Figura 2.7: Errores de Gradient Boosted Decision Tree

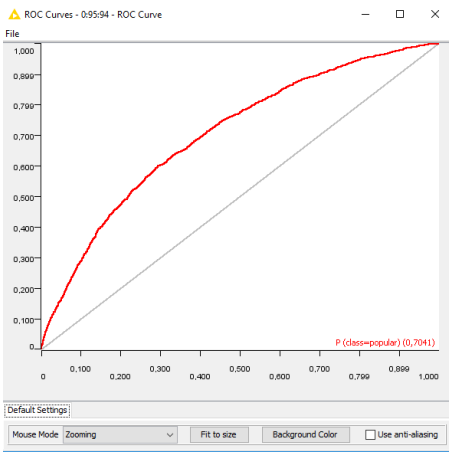

Figura 2.8: Curva ROC de Gradient Boosted Decision Tree

Medida	Valor
Accuracy	0,779
TPR	0,097
FPR	0,022
TNR	0,97
FNR	0,903
AUC	0,715
F1-Score	0,165
G-Mean	0,308

Tabla 2.4: Resultados de Gradient Boosted Decision Tree

2.4. Naive Bayes

El cuarto algoritmo que usaremos es un clasificador de Bayes [1], éste asume que la presencia o ausencia de una característica particular no está relacionada con la presencia o ausencia de cualquier otra característica, dada la clase variable.

Un clasificador de Bayes considera que cada una de estas características contribuye de manera independiente a la probabilidad de que una noticia sea popular, independientemente de la presencia o ausencia de las otras características. El flujo de trabajo empleado en KNIME es el que se muestra en la figura 2.13
Dentro de los nodos usados para el algoritmo tenemos X-partitioner , que básicamente nos permite configurar el numero de validaciones (5 en nuestro caso) y el muestreo estratificado. Después tenemos los nodos del algoritmo , en primer lugar el Naive Bayes Learner (nodo encargado del entrenamiento) el cual crea un modelo bayesiano, y en se-

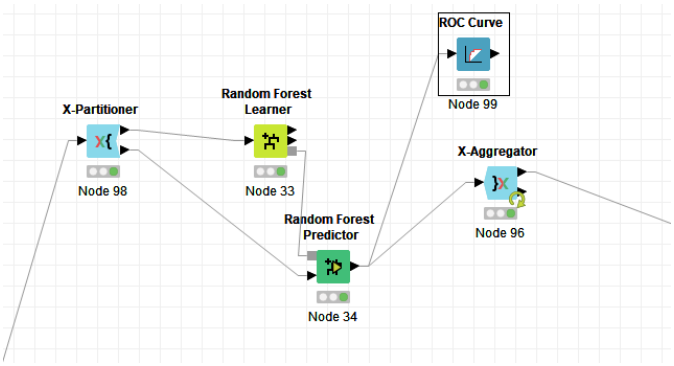

Figura 2.9: Workflow del algoritmo Random Forest

Row ID	popular	no_popular
popular	613	8313
no_popular	555	30163

Figura 2.10: Matriz de confisión de Random Forest

gundo lugar tenemos el nodo Naive Bayes Predictor , el cual predice la clase asociada según el modelo aprendido . Luego tenemos el nodo X-agregator , que se encarga de volver a reproducir los nodos anteriores como si un bucle se tratara, y también se encarga de sacar una tabla con las particiones y la tabla de errores. Por ultimo usamos el nodo Scorer , que nos permite mostrar la matriz de confusión y las estadísticas del algoritmo.

A continuación se muestra el siguiente contenido obtenido por Naive Bayes:
Una imagen con la matriz de confusión. (Figura 2.14)
Una imagen con la tabla de errores. (Figura 2.15)
Una imagen para la curva ROC. (Figura 2.16)
Una tabla con los datos resultantes de la ejecución del algoritmo (Tabla 2.6)

Row ID	Error in %	Size of Test Set	Error Count
fold 0	22.361	7929	1773
fold 1	22.374	7929	1774
fold 2	22.273	7929	1766
fold 3	22.411	7929	1777
fold 4	22.427	7928	1778

Figura 2.11: Errores de Random Forest

Medida	Valor
Accuracy	0,776
TPR	0,069
FPR	0,018
TNR	0,982
FNR	0,93
AUC	0,7
F1-Score	0,121
G-Mean	0,259

Tabla 2.5: Resultados de Random Forest

Medida	Valor
Accuracy	0,564
TPR	0,702
FPR	0,018
TNR	0,524
FNR	0,93
AUC	0,645
F1-Score	0,42
G-Mean	0,606

Tabla 2.6: Resultados de Naive Bayes

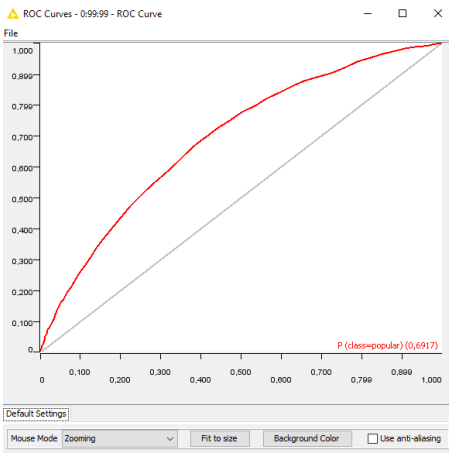

Figura 2.12: Curva ROC de Random Forest

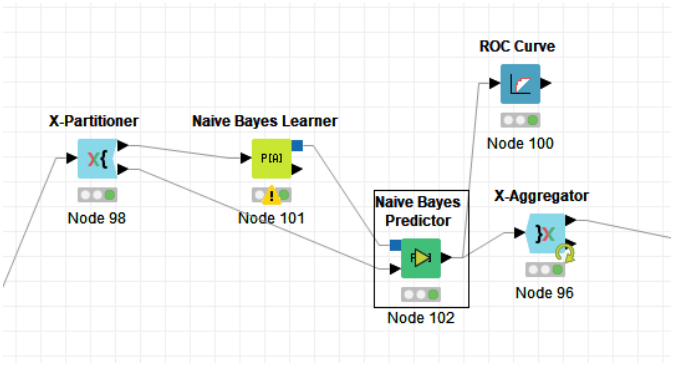

Figura 2.13: Workflow del algoritmo Naive Bayes

2.5. K-NN

El quinto algoritmo clasificador que usaremos es el de los N vecinos mas cercanos k-NN [9] . Básicamente calcula las distancias con todos los miembros de la colección y se toma un número K que son los que tienen la menor distancia. La mejor elección de k depende fundamentalmente de los datos y generalmente, valores grandes de k reducen el efecto de ruido en la clasificación, pero crean límites entre clases parecidas.

El flujo de trabajo empleado en KNIME es el que se muestra en la figura 2.17
Dentro de los nodos usados para el algoritmo tenemos X-partitioner , que basicamente nos permite configurar el numero de validaciones (5 en nuestro caso) y el muestreo estratificado. Después tenemos el nodo del algoritmo , nodo con dos entradas , una para el aprendizaje y otra para los test , y una única salida. La parte de pre-procesado es necesaria para obtener un buen modelo tras el funcionamiento del algoritmo , ya que ignora los registros que tienen valores perdidos y las variables numéricas tienen que estar

Row ID	popular	no_popular
popular	6263	2663
no_popular	14634	16084

Figura 2.14: Matriz de confisión de Naive Bayes

Row ID	Error in %	Size of Test Set	Error Count
fold 0	43.99	7929	3488
fold 1	44.495	7929	3528
fold 2	44.419	7929	3522
fold 3	41.821	7929	3316
fold 4	43.428	7928	3443

Figura 2.15: Errores de Naive Bayes

normalizadas. Pero para esta primera ejecución no se ha realizado el pre-procesado, éste se realizara en las secciones posteriores. Luego tenemos el nodo X-agregator , que se encarga de volver a reproducir los nodos anteriores como si un bucle se tratara, y cambien se encarga de sacar una tabla con las particiones y la tabla de errores. Por ultimo usamos el nodo Scorer , que nos permite mostrar la matriz de confusión y las estadísticas del algoritmo.

A continuación se muestra el siguiente contenido obtenido por K-NN:
Una imagen con la matriz de confusión. (Figura 2.18)
Una imagen con la tabla de errores. (Figura 2.19)
Una imagen para la curva ROC. (Figura 2.20)
Una tabla con los datos resultantes de la ejecución del algoritmo (Tabla 2.7)

Medida	Valor
Accuracy	0,77
TPR	0,045
FPR	0,016
TNR	0,979
FNR	0,955
AUC	0,554
F1-Score	0,798
G-Mean	0,209

Tabla 2.7: Resultados de K-NN

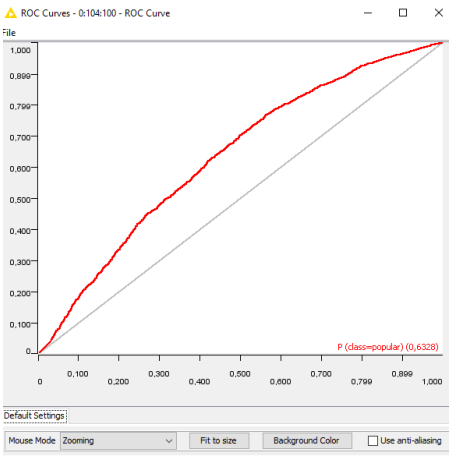

Figura 2.16: Curva ROC de Naive Bayes

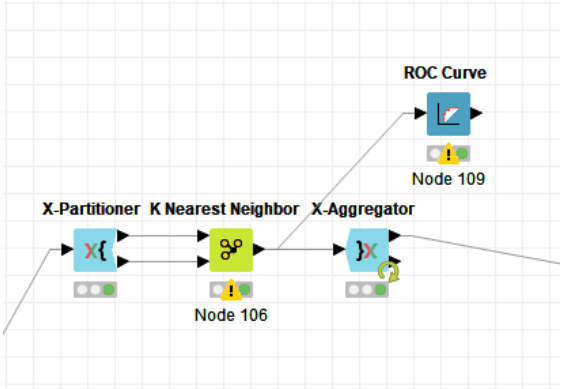

Figura 2.17: Workflow del algoritmo K-NN

2.6. Red Neuronal

El sexto algoritmo que usamos es el de Red Neuronal [10] , un modelo computacional basado en un gran conjunto de unidades neuronales simples. Cada neurona está conectada a otras , pudiendo incrementar los enlaces entre ellas el estado de activación de las adyacentes.

Al igual que para el caso de de K-NN , para poder ejecutar el algoritmo se requiere que las columnas categóricas con datos de tipo string sean transformadas a columnas numéricas , y que sean tratados los valores perdidos.

El flujo de trabajo empleado en KNIME es el que se muestra en las figuras 2.21 y 2.22.

La figura 2.21 muestra el contenido del metanodo del pre-procesado , que consiste en un

Row ID	popular	no_popular
popular	305	6540
no_popular	494	23228

Figura 2.18: Matriz de confisión de K-NN

Row ID	Error in %	Size of Test Set	Error Count
fold 0	39.967	7929	3169
fold 1	41.077	7929	3257
fold 2	40.875	7929	3241
fold 3	39.917	7929	3165
fold 4	41.36	7928	3279

Figura 2.19: Errores de K-NN

nodo para rellenar los valores perdidos , otro para filtrar la columna que es un string , y otro nodo para normalizar los valores numéricos.
La figura 2.22 muestra el contenido del algoritmo de Red Neuronal. Aparte de los nodos para realizar la validación cruzada , tenemos el nodo Rprop MLP Learner , que implementa el algoritmo Rprop para redes feedforward multicapa. A la derecha de este nodo tenemos el nodo MultiLayer Perception Predictor, que en base a el modelo proporcionado , calcula los valores de salida.
A continuación se muestra el siguiente contenido obtenido por Red Neuronal:
Una imagen con la matriz de confusión. (Figura 2.23)
Una imagen con la tabla de errores. (Figura 2.24)
Una imagen para la curva ROC. (Figura 2.25)
Una tabla con los datos resultantes de la ejecución del algoritmo (Tabla 2.8)

Medida	Valor
Accuracy	0,777
TPR	0,055
FPR	0,012
TNR	0,987
FNR	0,945
AUC	0,695
F1-Score	0,1
G-Mean	0,223

Tabla 2.8: Resultados de Red Neuronal

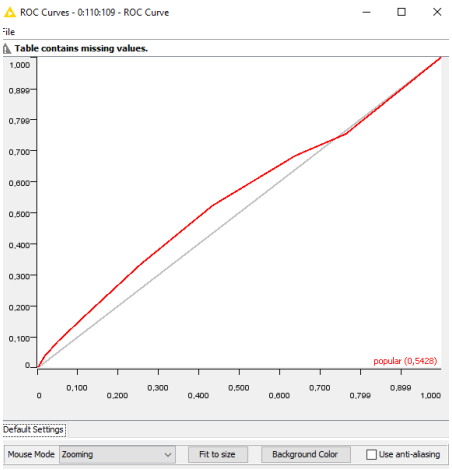

Figura 2.20: Curva ROC de K-NN

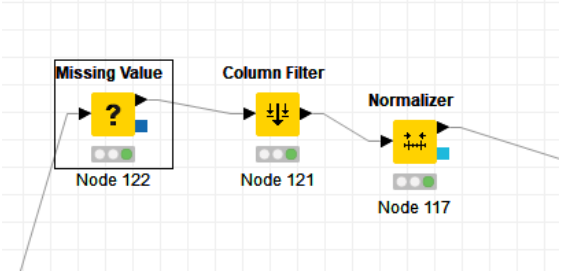

Figura 2.21: Workflow del preprocesado para la Red Neuronal

3. Análisis de resultados.

En este apartado se va a proceder al análisis comparativo de los distintos algoritmos teniendo en cuenta la tabla final de resultados, la gráfica ROC , y las distintas gráficas dispuestas para en análisis. Para poder realizar el análisis se han obtenido los resultados de cada algoritmo , y estos se han ido uniendo dos a dos con diferentes nodos Joiner para poder crear un metanodo que contenga los resultados de todos. El contenido del metanodo es el que se muestra en la figura 3.1 , y partimos desde este punto para la generación de visualizaciones.

En primer lugar tenemos la Figura final de resultados de todos los algoritmos juntos . La Figura 3.2
Y en segundo lugar tenemos la imagen de la comparativa entre las distintas curvas ROC. A partir de la tabla de resultados y de la gráfica ROC se pueden intuir varias cosas, la primera y más importante es que los algoritmos que mejores resultados obtuvieron son el GradientBoosted , el RandomForest y el de Red Neuronal, un poco peor se encuentra el NaiveBayes y los dos que obtuvieron peores resultados fueron el DecisionTree y el K-NN.

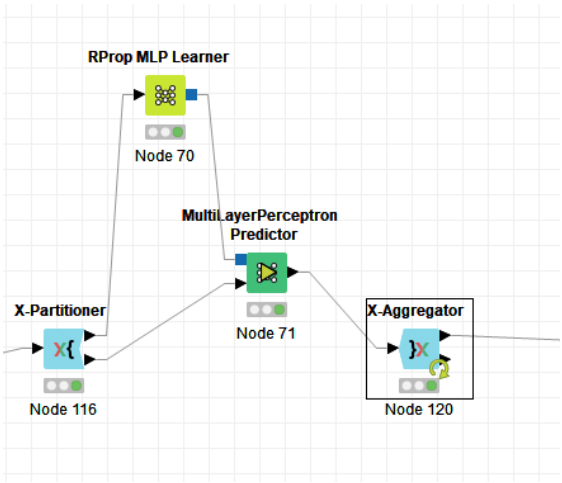

Figura 2.22: Workflow del algoritmo de Red Neuronal

Row ID	popular	no_popular
popular	493	8433
no_popular	392	30326

Figura 2.23: Matriz de confisión de Red Neuronal

La existencia de la diferencia entre los tres mejores y el Decision Tree es esperada, ya que estos tres primeros son bastante mas completos. Mas completos desde el punto de vista de que por ejemplo Gradient Boosted crea diferentes arboles de los cuales va corrigiendo errores conforme va aprendiendo de los anteriores, permitiendo crear un modelo mas estable que un solo árbol de decisión. También algo que les beneficia a GradientBoosted y a RandomForest es que son menos propensos al sobre-ajuste en en entrenamiento.

El algoritmo de Red Neuronal es ideal para problemas complejos, pero con el se obtiene un modelo de caja negra , tiene mayor carga computacional y es propenso al sobre-ajuste. En este primer caso se ha obtenido un modelo bastante bueno , estando a la par con RandomForest.

Row ID	Error in %	Size of Test Set	Error Count
fold 0	22.008	7929	1745
fold 1	22.399	7929	1776
fold 2	22.474	7929	1782
fold 3	22.184	7929	1759
fold 4	22.238	7928	1763

Figura 2.24: Errores de Red Neuronal

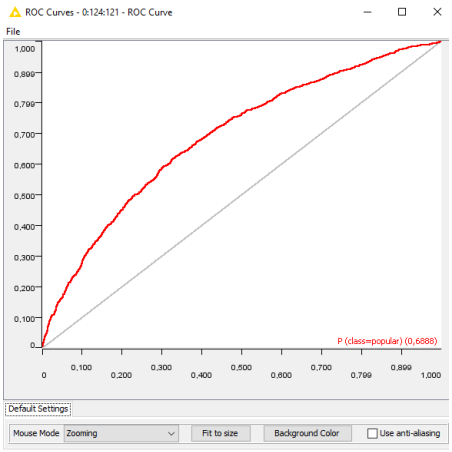

Figura 2.25: Curva ROC de Red Neuronal

Por lo que respecta al pesimo resultado de K-NN , se puede deber a su configuración por defecto , ya que al tener en cuenta solamente a los 3 vecinos mas cercanos comenzaría su ejecución estando más penalizado que los demás algoritmos.

En cuanto a los resultados obtenidos de los seis algoritmos respecto al Recall o TPR se tiene la figura 3.4. En ella se tienen en cuenta las variables de verdaderos positivos y falsos negativos, que son las que intervienen en el calculo del Recall. En esta gráfica se puede apreciar porque unos algoritmos tienen mejor Recall que otros. En primer lugar empezamos con el algoritmo Naive Bayes , el que mejor Recall tiene, tiene muchos verdaderos positivos y pocos falsos negativos. A este algoritmo le sigue el de Decision Tree , pero con unos valores bastante inferiores. Por ultimo tenemos a los demás algoritmos , que tienen unos valores pésimos muy similares entre ellos. De éstos cuatro últimos se podría decir que tienen un mal comportamiento en su configuración por defecto.

En cuanto a los resultados obtenidos de los seis algoritmos respecto al Specifity o TNR se tiene la figura 3.5. En ella se tienen en cuenta las variables de falsos positivos y verdaderos negativos, que son las que intervienen en el calculo del Specifity. En esta gráfica se puede apreciar porque unos algoritmos tienen mejor Specifity que otros. En primer lugar se aprecia como Red Neuronal , K-NN , Random Forest y Gradient Boosted tienen los mejores resultados de Specifity . Al contrario que con el Recall , el algoritmo Naive Bayes se posiciona en último lugar , por lo que tenemos con Naive Bayes un modelo un poco 'indeciso' debido al gran aumento de falsos negativos. También merece la pena ver como el algoritmo Red Neuronal esta en primera posición.

Lo que se puede observar con las gráficas 3.4 y 3.5 es que algoritmos son regulares al Recall y Specifity. Por lo que sus modelos (en su configuración por defecto) se adaptan bien a nuestro tipo concreto de datos.

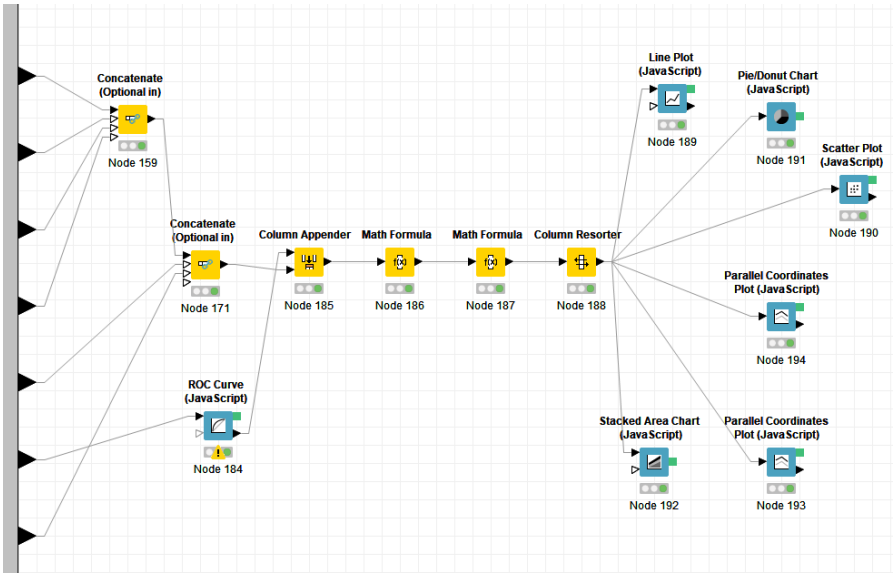

Figura 3.1: Metanodo para la generacion de visualizaciones

Row ID	Area Under Curve	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Specifity	F-measure	nombre	G-mean	Accuracy
Tree Decision	0.548	2465	5261	25450	6456	0.276	0.319	0.829	0.296	Tree Decision	0.479	0.704
Gradient Boos...	0.715	865	697	30021	8061	0.097	0.554	0.977	0.165	Gradient Bo...	0.308	0.779
Naive Bayes	0.645	6263	14634	16084	2663	0.702	0.3	0.524	0.42	Naive Bayes	0.606	0.564
Random Forest	0.7	613	555	30163	8313	0.069	0.525	0.982	0.121	Random For...	0.26	0.776
K-NN	0.554	305	494	23228	6540	0.045	0.382	0.979	0.08	K-NN	0.209	0.77
Red Neuronal	0.695	493	392	30326	8433	0.055	0.557	0.987	0.1	Red Neuronal	0.234	0.777

Figura 3.2: Resultados de los diferentes algoritmos

Para finalizar esta comparativa vamos a ver en la figura 3.6 el comportamiento de los algoritmos en cuanto a su Accuracy se refiere. Se puede apreciar que los 4 algoritmos dominantes son GradientBoosted, RandomForest, K-NN Y Red Neuronal. Dejando a NaiveBayes 'por los suelos' .
Para finalizar decimos que los algoritmos GradientBoosted , RandomForest y Red Neuronal son los que mejor se están comportando en nuestro estudio con las configuraciones por defecto.

4. Configuración de algoritmos.

En este apartado vamos a comentar los distintos parámetros y configuraciones usadas a lo largo de la práctica. A su vez se mostrará para algunos algoritmos una configuración diferente y se compararán los resultados, en donde podremos ver aspectos como el sobre aprendizaje y la obtención de modelos más simples

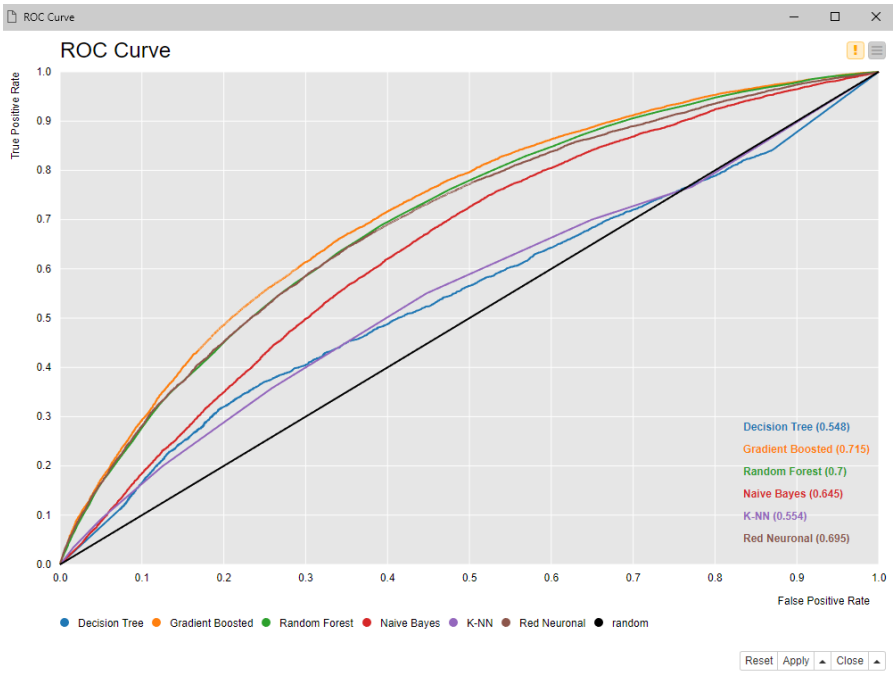

Figura 3.3: Curvas ROC de los diferentes algoritmos

4.1. Decision Tree

En primer lugar tenemos el árbol de decisión con sus valores por defecto, en este caso los parámetros de configuración son los del nodo Decision Tree Learner. En la figura 4.1 se puede apreciar la columna de la clase, la cuál será siempre la misma. Como medida de calidad está indicado el Gini Index y no hay método de poda seleccionado. En cuanto al mínimo número de registros por nodo viene por defecto en 2. Y el número de registros para almacenar por vista en 10000.

Para intentar sacar otro modelo vamos a aumentar el número mínimo de registros por nodo, de 2 a 10. La configuración es la que se muestra en la figura 4.2 .Con esto podemos estar ocasionando pérdida de precisión, pero si ocurre que la pérdida es poca, podríamos estar ante una mejor solución, por la mayor simplicidad del modelo. En cuanto al número de registros para almacenar por vista lo dejaremos en 10000.
Para comparar los resultados extraemos una tabla con los resultados de ambas configuraciones (figura 4.3) , así como sus gráficas ROC para ver sus curvas (figura 4.4).

En este caso se puede ver como el modelo con la configuración por defecto estaba sobre aprendiendo , de forma que se sobre adaptaba a los datos de entrenamiento y con los datos de test no cumplía con las expectativas. La forma mas clara de apreciarlo es que simplificando el modelo se tienen mejores resultados. Por lo que estamos obteniendo un modelo más simplificado y ganando precisión.

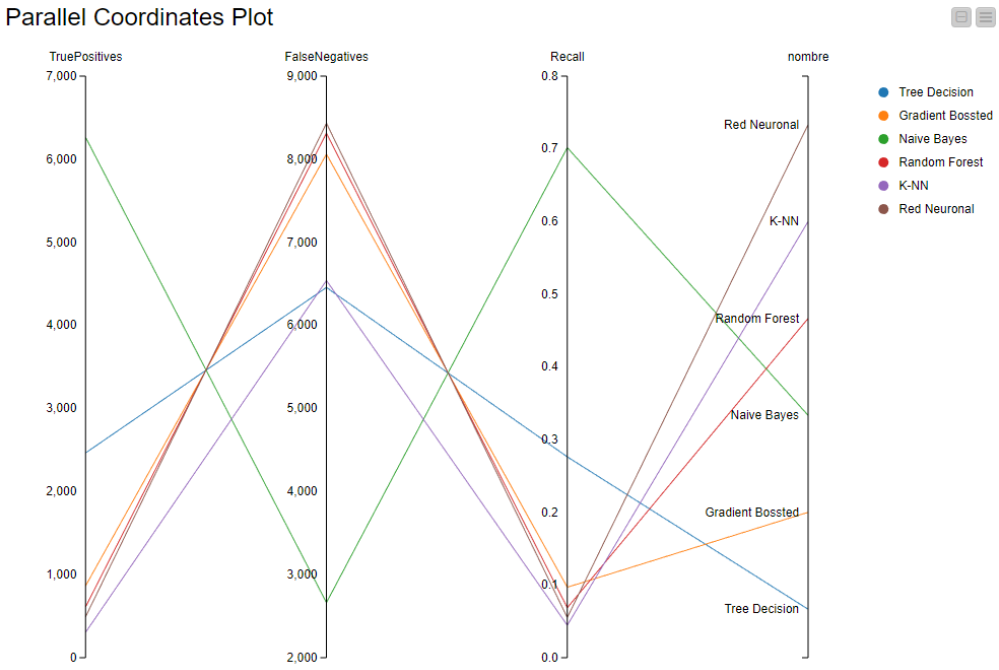

Figura 3.4: Grafica del Recall de los 6 algoritmos , integrando tanto verdaderos positivos como falsos negativos

4.2. Gradient Boosted

Pasando al segundo algoritmo, vamos a ver los parámetros que tiene por defecto. Parámetros como la selección de atributos, numero limite de niveles (profundidad) , numero de modelos o tasa de aprendizaje. La figura 4.5 muestra la configuración por defecto, donde tenemos el límite de profundidad del árbol en 4.
Para intentar sacar un modelo más preciso y ver las diferencias entre ambos modelos , vamos a aumentar el límite de profundidad del árbol de 4 a 10 , e incrementar el numero de modelos de 100 a 200, como muestra la figura 4.6.
Para comparar los resultados extraemos una tabla con los resultados obtenidos de ambas configuraciones (figura 4.7) , así como sus gráficas ROC para ver sus curvas (figura 4.8).

En este caso , vemos como aumentando la complejidad y el tiempo de ejecución del modelo no conseguimos mejorar sustancialmente los resultados, sino que hemos perdido Precisión y Specifity, provocado por el aumento de falsos positivos.
En contraposición nos encontramos con un Recall en aumento en el segundo algoritmo, ya que se distancia con respecto los otros dos en verdaderos positivos, siendo los falsos negativos prácticamente similares a los demás algoritmos.
Con respecto al valor del AUC, apenas se nota la diferencia entre las curvas .

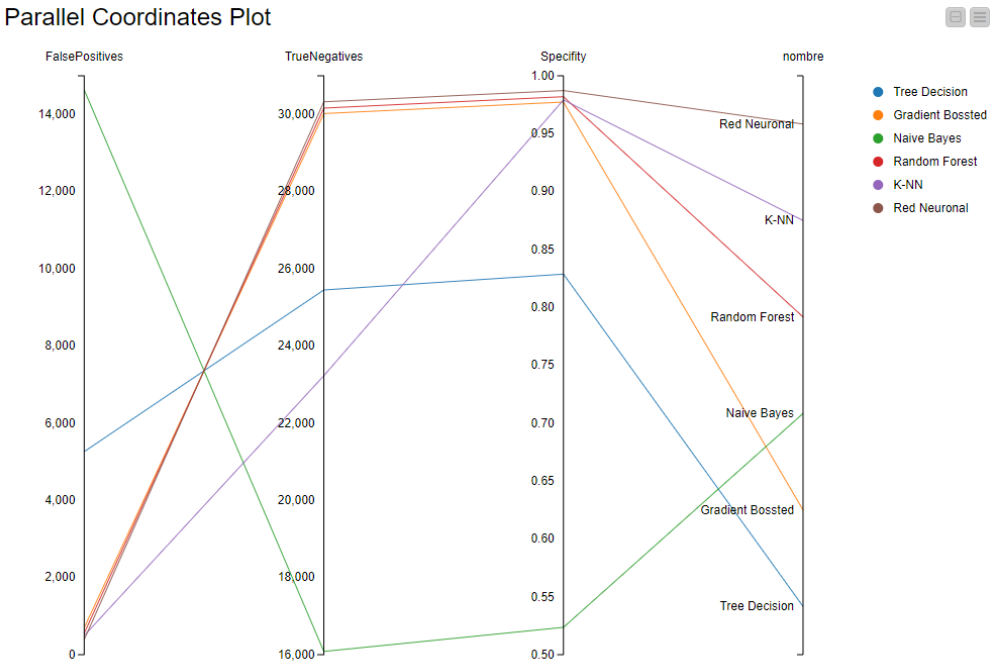

Figura 3.5: Grafica del Specifity de los 6 algoritmos , integrando tanto falsos positivos como verdaderos negativos

4.3. Naive Bayes

Para este algoritmo únicamente se han realizado las ejecuciones con los parámetros por defecto. Los parámetros por defecto son : La columna de clasificacion (class), probabilidad por defecto (0.0) , máximo numero de valores nominales únicos por atributo (100), ignorar valores perdidos y crear un modelo compatible PMML (ambos desmarcados). La figura 4.9 muestra esta configuración.
Tanto los resultados como la curva ROC obtenida por este algoritmo se han mostrado con anterioridad en la sección 2.4.

4.4. Random Forest

En el caso del algoritmo Random Forest , al igual que en el caso anterior con Naive Bayes, únicamente se han realizado pruebas con los parámetros por defecto. Los parámetros que nos encontramos son los siguientes: Columnas de atributos que añadimos (todos) , patrones a almacenar (2000) , criterio de división (gain ratio) , profundidad del árbol (10), mínimo tamaño del nodo (1), y numero de modelos (100) . La figura 4.10 muestra esta configuración.
Tanto los resultados como la curva ROC obtenida por este algoritmo se han mostrado con anterioridad en la sección 2.3.

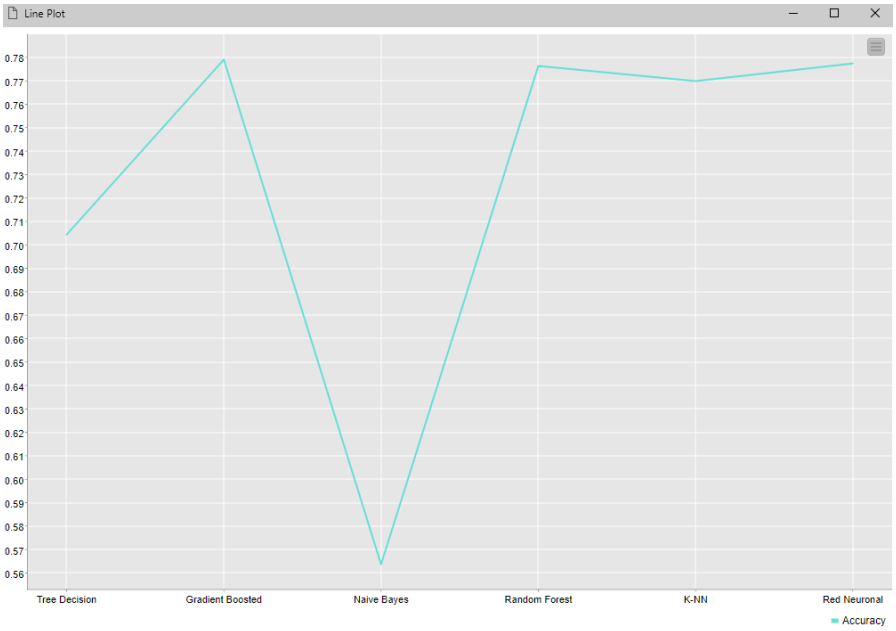

Figura 3.6: Grafica del Accuracy de los 6 algoritmos

4.5. K-NN

Para éste algoritmo se han realizado pruebas con dos configuraciones diferentes.
Una primera configuración con los siguientes parámetros : Columna con la clase (class) , el numero de vecinos a considerar (10) , peso de esos vecinos por distancia (off) y salida de las probabilidades de la clase , que se usa para sacar la gráfica ROC. Esta configuración es la que se representa con la figura 4.11
La segunda configuración que usamos cuenta con los siguientes parámetros : Columna con la clase (class) , el numero de vecinos a considerar (30) , peso de esos vecinos por distancia (on, incluye la distancia del patrón de consulta a los patrones de entrenamiento , de tal forma que los vecinos mas cercanos tendrán una mayor influencia) y salida de las probabilidades de la clase , que se usa para sacar la gráfica ROC. Esta configuración es la que se representa con la figura 4.12
Para comparar los resultados , extraemos una tabla con los resultados de ambas configuracienes. (figura 4.13)
En las dos modificaciones que se han realizado se puede observar que no existe sobre aprendizaje, ya que al aumentar el numero de vecinos y tener en cuenta los pesos , los resultados no mejoran , sino todo lo contrario , empeoran. Disminuye el numero de verdaderos positivos y aumenta el de falsos negativos. Cabe destacar que mejoran la Precision y el Specifity respecto a la primera configuración con 10 vecinos.

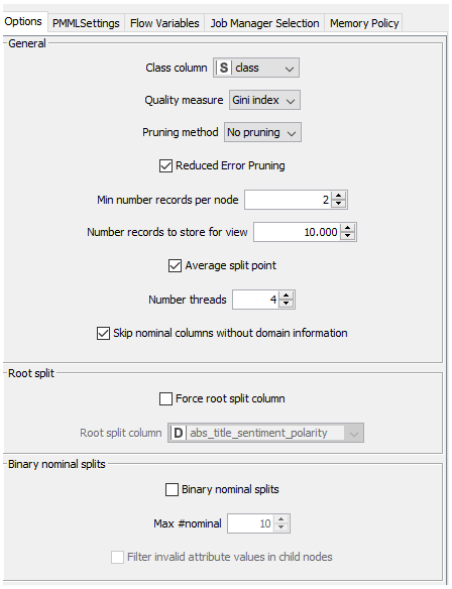

Figura 4.1: Configuración por defecto del algoritmo Decision Tree

4.6. Red Neuronal

Por último, tenemos el algoritmo Red Neuronal en donde se han realizado una comparación como en los casos anteriores, modificando los ajustes y parámetros de configuración propios del algoritmo.

En cuanto a la configuración por defecto tenemos, parámetro de máximo numero de iteraciones (100) , numero de capas ocultas (1), numero de neuronas ocultas por capa (10), ignorar valores perdidos (on) y semilla (mi DNI). Configuración que se muestra en la figura 4.14
En lo que se refiere al modelo modificado de Red Neuronal , se ha modificado el numero de capas ocultas de 1 a 15, y el numero de neuronas ocultas de cada capa a 20. Esto provoca un modelo mas complejo , que no puede visualizarse y por lo tanto sacar conclusiones claras de el como por ejemplo un árbol de decisión. Esta configuración es la que se muestra en la figura 4.15
Para comparar los resultados extraemos una tabla con los resultados obtenidos de ambas configuraciones (figura 4.16) , así como sus gráficas ROC para ver sus curvas (figura 4.17).

En los resultados obtenidos se puede apreciar la existencia de sobre aprendizaje , ya que en el primer caso con la configuración se obtienen mejores resultados que con un modelo mas complejo con mayor numero de capas ocultas y de neuronas por capa. El modelo por defecto tiene mejor Precision , Specifity , y AUC (esta última se puede apreciar en la gráfica ROC).

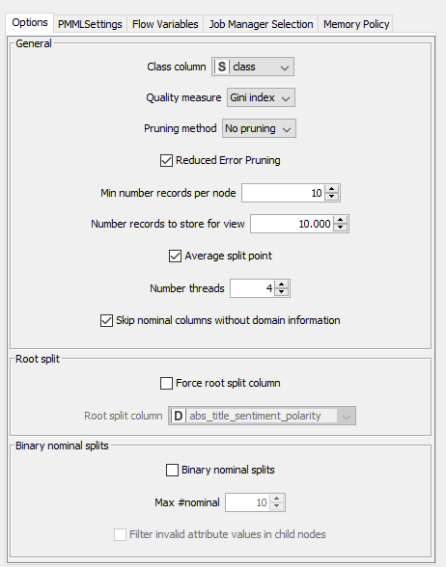

Figura 4.2: Configuración version 2.0 algoritmo Decision Tree

Row ID	I TruePositives	I FalsePositives	I TrueNegatives	I FalseNegatives	D Recall	D Precision	D Specifity	D F-measure
Tree Decision	2465	5261	25450	6456	0.276	0.319	0.829	0.296
Tree Decision 2.0	2161	4315	26403	6765	0.242	0.334	0.86	0.281

Figura 4.3: Tabla comparativa de las configuraciondes de Decision Tree

Este estudio de los resultados realizado se ve afectado por el pre procesamiento aleatorio de los datos. Procesamiento que se explico en secciones anteriores y que se modificara en distintas versiones posteriormente.

5. Procesado de datos.

En esta sección se habla de los distintos pre-procesados de datos que se han realizado para algunos de los algoritmos con la intención de mejorar los resultados que hemos obtenido hasta ahora, aunque en algunos casos no conseguimos nuestro objetivo de mejorar el modelo. Aun así es interesante conocer los distintos comportamientos de los algoritmos. Las funcionalidades de los nodos usados para el procesado de datos en KNIME han sido normalizar, filtrar, discretizar , etc...

Para nuestro caso nos hemos centrado en los algoritmos de Decision Tree, Naive Bayes y Random Forest. Estos dos últimos han sido elegidos por el hecho de que en la sección anterior no se realizo un estudio de las diferentes configuraciones posibles.

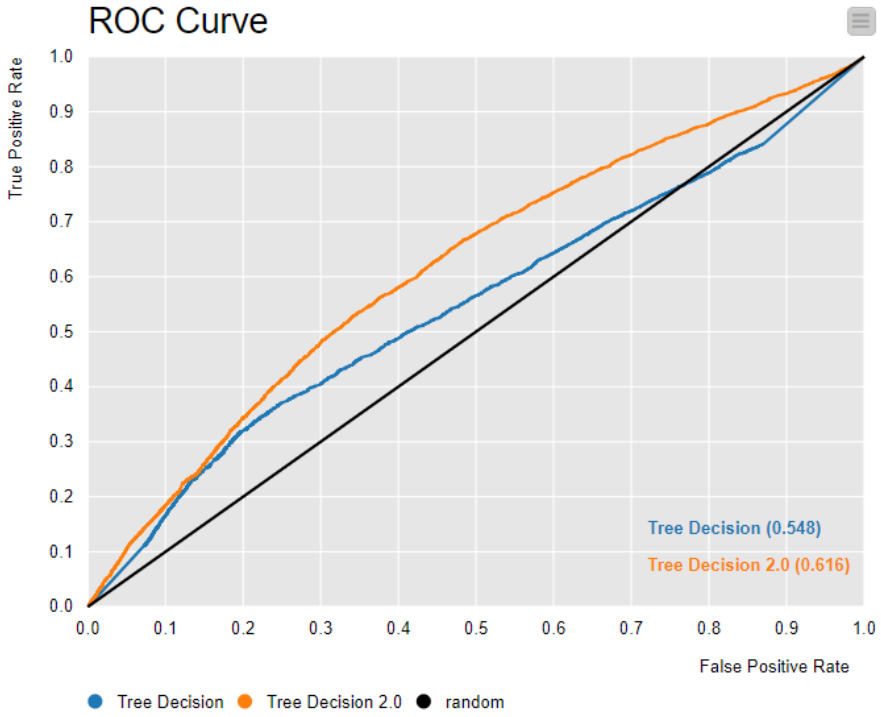

Figura 4.4: Graficas ROC de las configuraciondes de Decision Tree

5.1. Decision Tree

Para este algoritmo hemos considerado las dos versiones obtenidas del apartado 4.1 y le hemos aplicado el siguiente procesamiento:

One to Many: Transforma todos los posibles valores de una columna en una nueva columna. De tal forma que en las nuevas columnas sale 1 cuando la fila pertenece a ese valor, y 0 cuando no. El nodo agrega tantas columnas como posibles valores . En nuestro caso lo hacemos con la columna de weekday (String).

Column filter: Elimina las columnas que previamente han sido transformadas en nuevas columnas por el nodo One to Many. En nuestro caso eliminamos la columna weekday (String)

Missing Values: Maneja los valores perdidos de las distintas celdas, de tal forma que se puede actuar reponiendo valores en forma de media , mas frecuente o eliminar la fila. Para nuestro caso lo usaremos para las casillas numéricas , sustituyendo el valor perdido por la media, y para las string pondremos el valor mas frecuente.

Normalizer: Normaliza los valores de las columnas numéricas entre el rango que se desee.

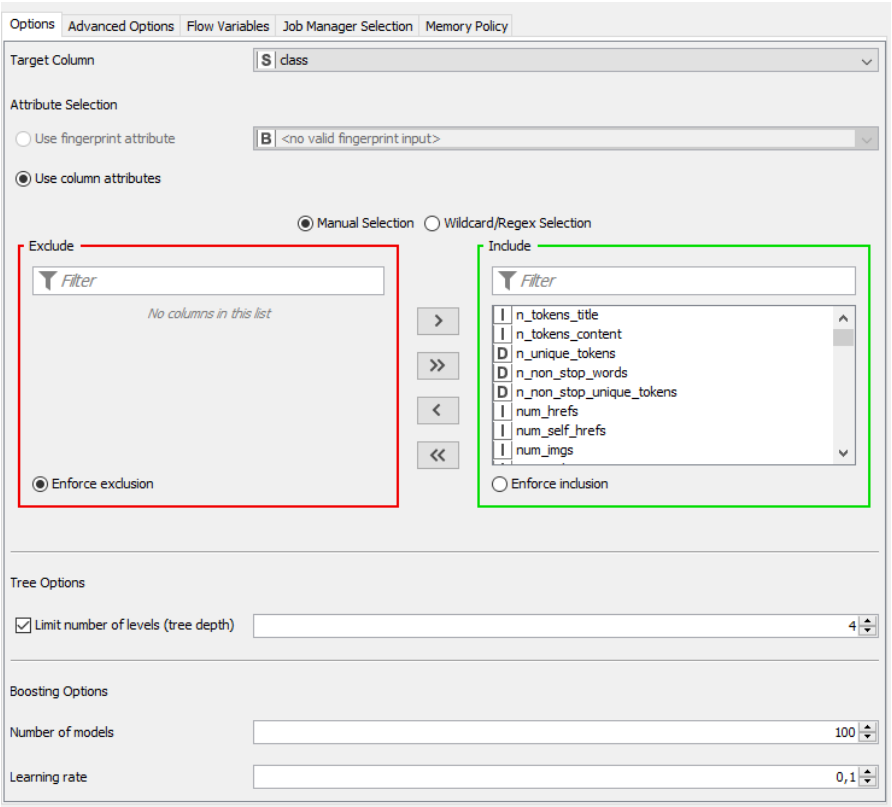

Figura 4.5: Configuración por defecto del algoritmo Gradient Boosted

Para nuestro caso entre 0 y 1.

A continuacion se muestra la figura 5.1 con el Workflow usado en KNIME para el procesamiento de los datos.
Para comparar los resultados obtenidos sin y con el procesado se muestra la figura 5.2 como tabla comparativa. Así como la figura 5.3 con las curvas ROC de los resultados obtenidos sin y con el procesado de datos.
Con este procesamiento se ha conseguido que el algoritmo trabaje sin columnas categóricas, eliminando las mismas y convirtiéndolas en nuevas columnas numéricas , sin valores perdidos y normalizando el conjunto de valores a una escala apropiada. A través de la comparativa de resultados y de las graficas ROC se puede deducir que con el procesado se consigue un mejor Recall , peor Precision y peor Specfifity. Respecto al AUC de cada par de algoritmos con y sin procesado no se aprecian diferencias significativas.

5.2. Random Forest

Para el algoritmo Random Forest hemos considerado 2 preprocesamientos diferentes:

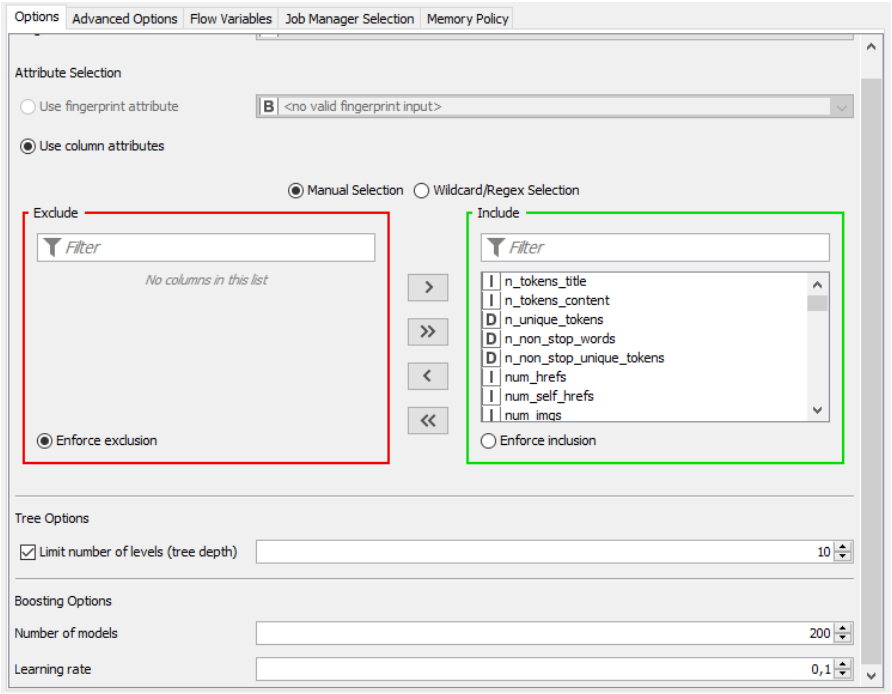

Figura 4.6: Configuración version 2.0 del algoritmo Gradient Boosted

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Specifity	F-measure
Gradient Bossted	865	697	30021	8061	0.097	0.554	0.977	0.165
Gradient Bossted 2.0	1264	1327	29391	7662	0.142	0.488	0.957	0.22

Figura 4.7: Tabla comparativa de las configuraciondes de Decision Tree

El primero es el mismo procesamiento que el utilizado en el apartado anterior con el algoritmo Decision Tree. Los nodos usados son One to Many, Column Filter, Missing Values y Normalicer, todos ellos con la misma configuracion . La figura 5.1 muestra el Workflow usado en KNIME para primer procesamiento de los datos.

A continuacion se muestra la figura 5.4 con el Workflow usado en KNIME para el segundo procesamiento de los datos, en el hemos usado los siguientes nodos:

Equal Size Sampling: Elimina filas aleatoriamente pertenecientes a la clase de la mayoria, de tal forma que devuelve todas las filas de la clase minoritaria y una muestra aleatoria de la clase mayoritaria, realizando un balanceo entre clases. (UnderSampling).

Missing Values: Maneja los valores perdidos de las distintas celdas, de forma que se puede actuar reponiendo valores de distintas formas. En nuestro caso lo usamos para sustituir los valores numericos perdidos por la media.

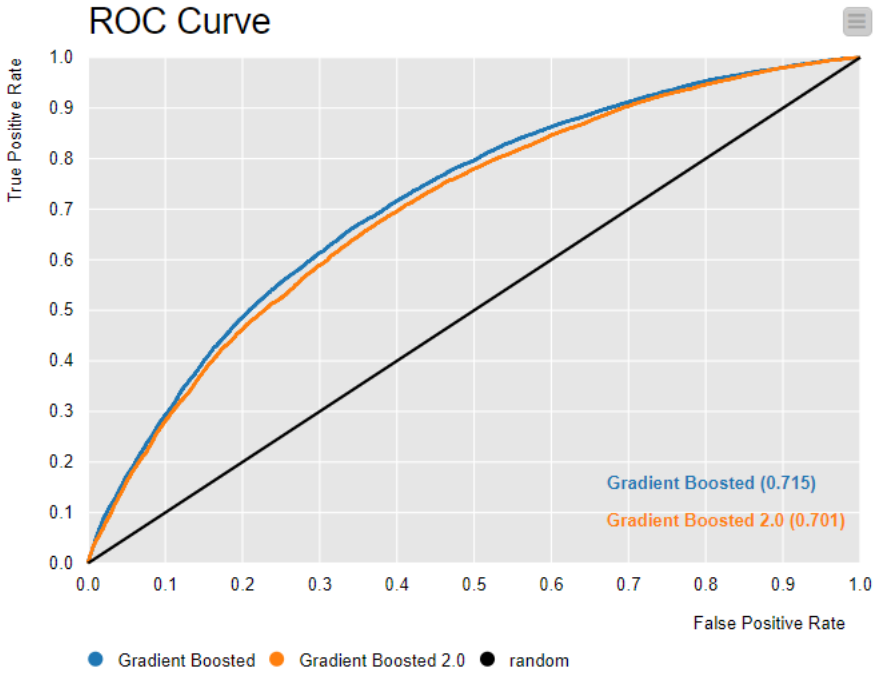

Figura 4.8: Graficas ROC de las configuraciondes de Decision Tree

Normalizer: Normaliza los valores de las columnas numéricas en el rango deseado , en nuestro caso usamos el rango entre 0 y 1 para todos los valores numericos

Para comparar los resultados obtenidos sin y con el procesado se muestra la figura 5.5 como tabla comparativa. Así como la figura 5.6 con las curvas ROC de los resultados obtenidos sin y con el procesado de datos.
Con este procesamiento se ha conseguido que el algoritmo trabaje sin columnas categóricas, eliminando las mismas y convirtiéndolas en nuevas columnas numéricas , sin valores perdidos y normalizando el conjunto de valores a una escala apropiada. A través de la comparativa de resultados y de las graficas ROC se puede deducir que con el procesado se consigue un mejor Recall , mejor Precision y peor Specfifity en el caso del preprocesado 2. Respecto al AUC de los algoritmos con y sin procesado no se aprecian diferencias.

5.3. Naive Bayes

Para este algoritmo hemos considerado 2 preprocesamientos diferentes.

En primer lugar hemos usado el procesamiento común a los dos algoritmos anteriores (figura 5.1) y en segundo lugar hemos usado el procesamiento de la subseccion anterior (figura 5.4).

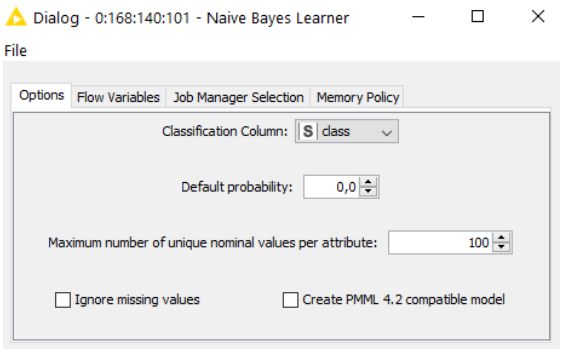

Figura 4.9: Configuración por defecto del algoritmo Naive Bayes

Para comparar los resultados obtenidos sin y con el procesado se muestra la figura 5.7 como tabla comparativa. Así como la figura 5.8 con las curvas ROC de los resultados obtenidos sin y con el procesado de datos.
Con este procesamiento se ha conseguido que el algoritmo trabaje sin columnas categóricas, eliminando las mismas y convirtiéndolas en nuevas columnas numéricas , sin valores perdidos y normalizando el conjunto de valores a una escala apropiada. A través de la comparativa de resultados y de las gráficas ROC se puede deducir que con el procesado se consigue un mejor Recall , mejor Precision y peor Specfifity. Respecto al AUC de los algoritmos con y sin procesado no se aprecian diferencias (se observa en la grafica ROC).

6. Interpretación de resultados.

7. Bibliografía.

Referencias

[1] Clasificador bayesiano
https://es.wikipedia.org/wiki/Clasificador_bayesiano_ingenuo

[2] Clasificador de bayes
https://nlp.stanford.edu/IR-book/html/htmledition/
naive-bayes-text-classification-1.html

[3] onlinespopularity
https://sci2s.ugr.es/sites/default/files/files/Teaching/
GraduatesCourses/InteligenciaDeNegocio/Curso18-19/
onlinenewspopularity.xlsx

[4] Decision Tree
https://es.wikipedia.org/wiki/%C3%81rbol_de_decisi%C3%B3n

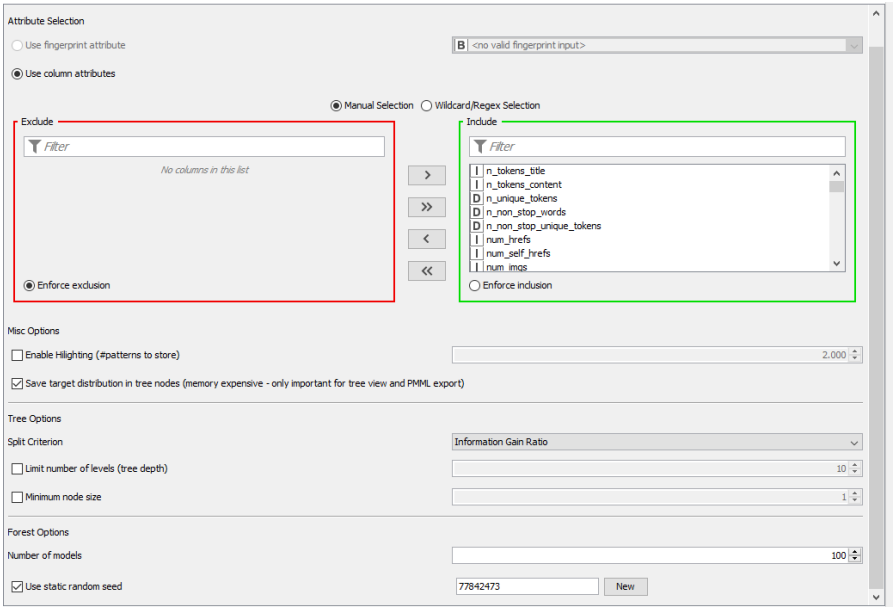

Figura 4.10: Configuración por defecto del algoritmo Random Forest

[5] Decision Tree
https://www.gestiondeoperaciones.net/procesos/arbol-de-decision/

[6] Gradient boosting
https://en.wikipedia.org/wiki/Gradient_boosting

[7] Gradient boosting
https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

[8] Random forest
https://es.wikipedia.org/wiki/Random_forest

[9] K-NN
https://es.wikipedia.org/wiki/K_vecinos_m%C3%A1s_pr%C3%B3ximos

[10] Red Neuronal
https://es.wikipedia.org/wiki/Red_neuronal_artificial

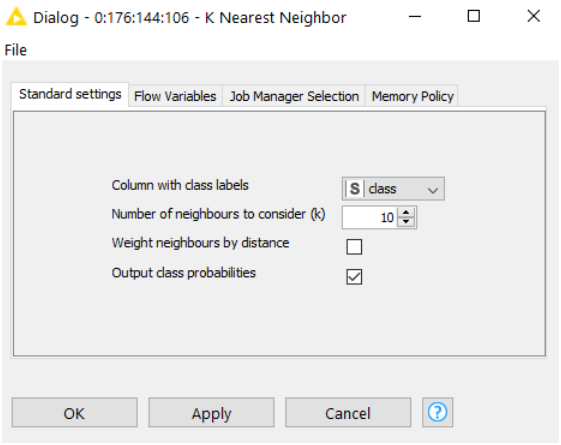

Figura 4.11: Configuración 1 del algoritmo K-NN

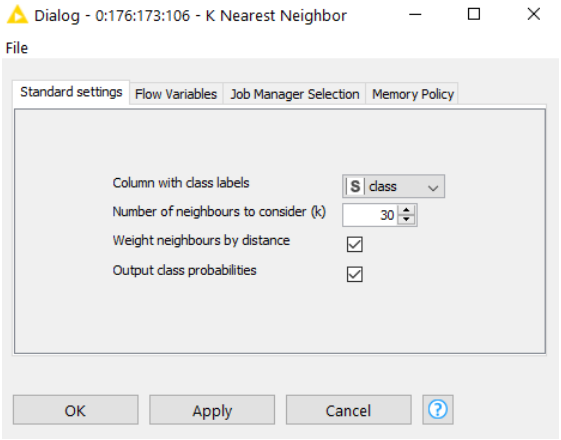

Figura 4.12: Configuración 2 del algoritmo K-NN

Row ID	I TruePositives	I FalsePositives	I TrueNegatives	I FalseNegatives	D Recall	D Precision	D Specifity	D F-measure
K-NN	305	494	23228	6540	0.045	0.382	0.979	0.08
K-NN 2.0	101	125	23597	6744	0.015	0.447	0.995	0.029

Figura 4.13: Tabla comparativa de las configuraciondes de K-NN

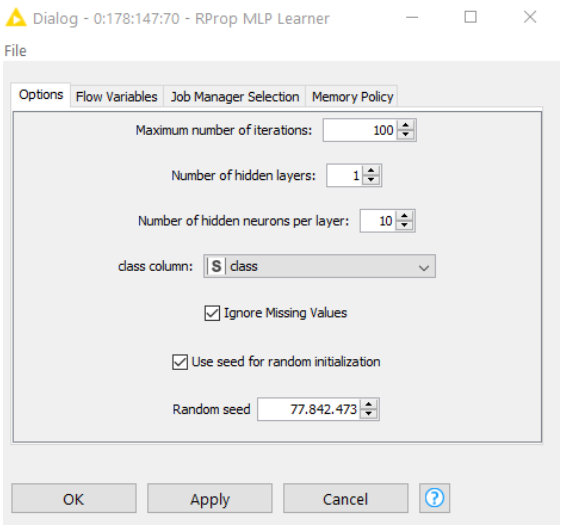

Figura 4.14: Configuración por defecto del algoritmo Red Neuronal

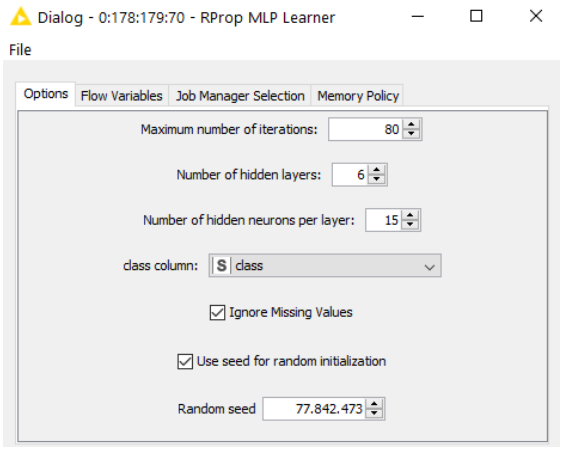

Figura 4.15: Configuración modificada del algoritmo Red Neuronal

Row ID	I TruePositives	I FalsePositives	I TrueNegatives	I FalseNegatives	D Recall	D Precision	D Specifity	D F-measure
Red Neuronal	493	392	30326	8433	0.055	0.557	0.987	0.1
Red Neuronal 2.0	838	772	29946	8088	0.094	0.52	0.975	0.159

Figura 4.16: Tabla comparativa de las configuraciondes de Red Neuronal

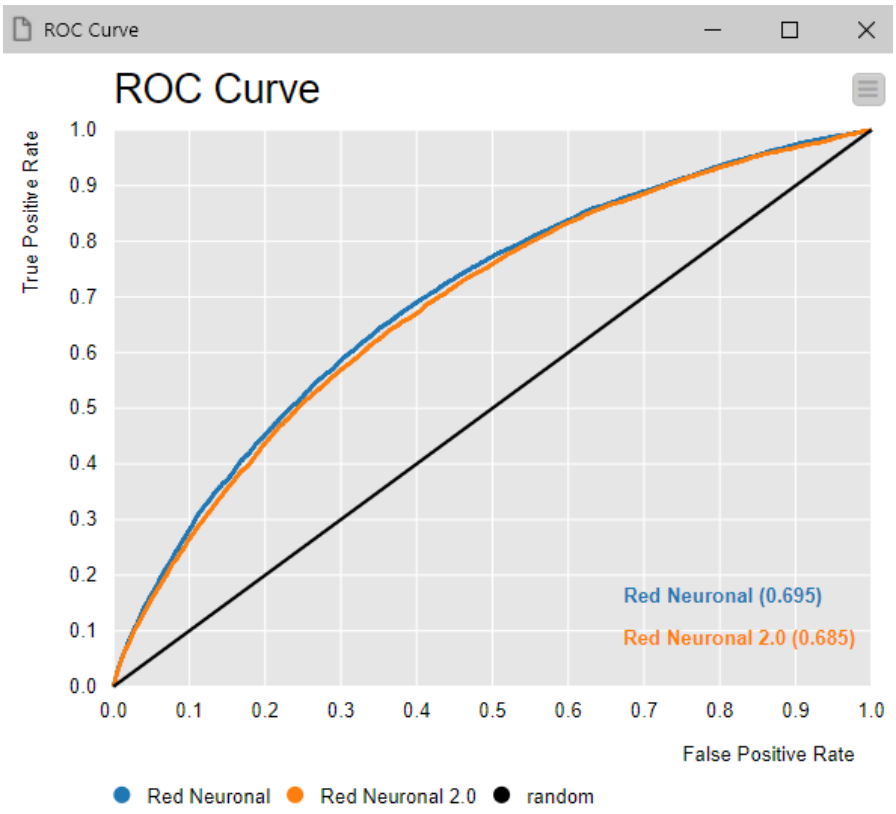

Figura 4.17: Graficas ROC de las configuraciondes de Red Neuronal

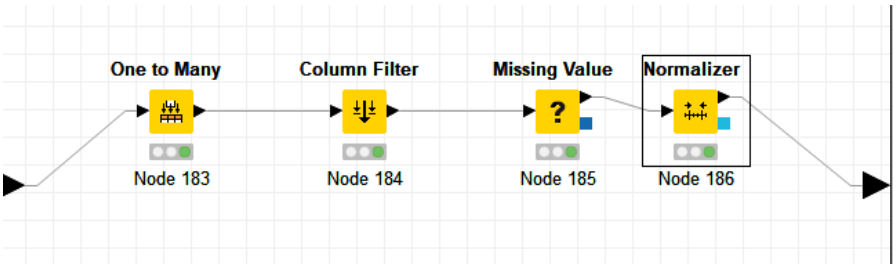

Figura 5.1: Workflow en Knime para el procesado de Decision Tree

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Specifity	F-measure
Tree Decision	2465	5261	25450	6456	0.276	0.319	0.829	0.296
Tree Decision 2.0	2161	4315	26403	6765	0.242	0.334	0.86	0.281
Tree Decision Procesado	2709	6259	24425	6204	0.304	0.302	0.796	0.303
Tree Decision 2.0 procesado	2334	4803	25915	6591	0.262	0.327	0.844	0.291

Figura 5.2: Comparativa de Resultados de Decision Tree, sin y con procesado

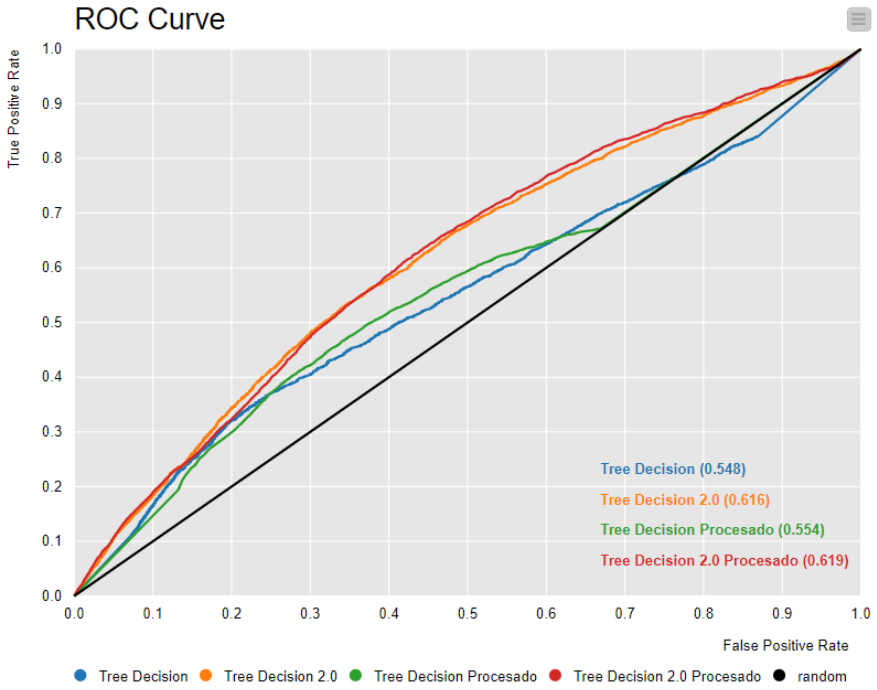

Figura 5.3: Comparativa de curva ROC de Decision Tree, sin y con procesado

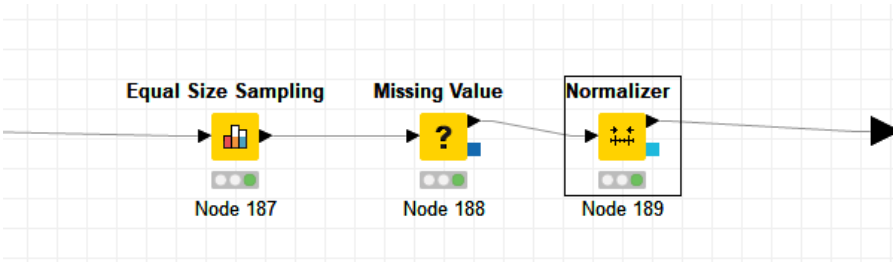

Figura 5.4: Workflow en Knime para el procesado 2 de Random Forest

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Specifity	F-meas...	nombre
Random Forest	613	555	30163	8313	0.069	0.525	0.982	0.121	Random Forest
Random Forest Procesado 1	635	574	30144	8291	0.071	0.525	0.981	0.125	Random Fore...
Random Forest Procesado 2	5839	3168	5758	3087	0.654	0.648	0.645	0.651	Random Fore...

Figura 5.5: Comparativa de Resultados de Random Forest, sin y con procesado

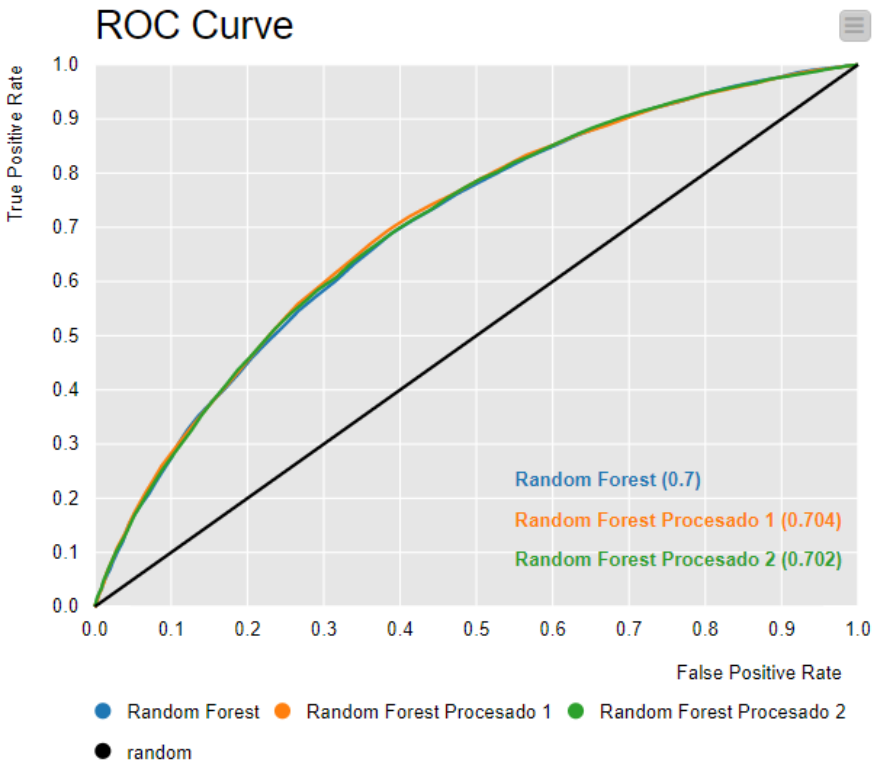

Figura 5.6: Comparativa de curva ROC de Random Forest, sin y con procesado

Row ID	I TruePositives	I FalsePositives	I TrueNegatives	I FalseNegatives	D Recall	D Precision	D Specifity	D F-measure
Naive Bayes	6263	14634	16084	2663	0.702	0.3	0.524	0.42
Naive Bayes Procesado 1	6359	14685	16033	2567	0.712	0.302	0.522	0.424
Naive Bayes Procesado 2	7391	5699	3227	1535	0.828	0.565	0.362	0.671

Figura 5.7: Comparativa de Resultados de Naive Bayes, sin y con procesado

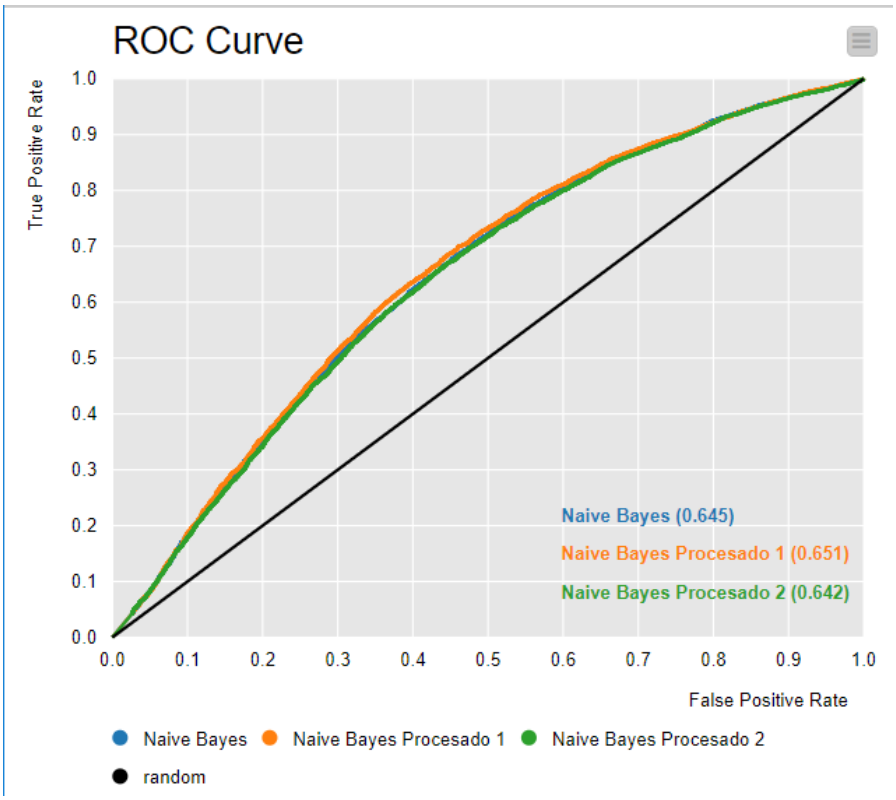

Figura 5.8: Comparativa de curva ROC de Naive Bayes, sin y con procesado