Universidad de Granada

Ingeniería Informática

Análisis Predictivo Empresarial Mediante Clasificación.

Autor: José Antonio Ruiz Millán
email: jantonioruiz@correo.ugr.es
Curso: 2018-2019
Asignatura: Inteligencia de Negocio
4 de noviembre de 2018

Índice

1. Introduccion

En esta practica se aborda el problema de la ***Predicción de la Popularidad de Noticias Online***.

La alta proliferación de espacios con noticias en la Web hace cada vez más útil la predicción automática de la popularidad de estas noticias. En esta práctica se propone un sistema inteligente de apoyo a la decisión (SIAD) que analiza los artículos antes de su publicación. Se usa un amplio conjunto de características extraídas (por ejemplo, palabras clave, contenido de medios digitales, popularidad anterior de las noticias a las que se hace referencia en el artículo, etc.). El SIAD predice si un artículo se volverá popular. En un caso real, esta herramienta puede combinarse con un algoritmo de optimización para mejorar las características del artículo y hacerlo así más popular. El conjunto de datos se basa en 39.644 noticias extraídas en 2015 de la web `http://mashable.com`. Además de las características extraídas sobre estas noticias, se conoce también el número de veces que la noticia ha sido compartida. A partir de este dato, 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.

La práctica consiste principalmente en el estudio del comportamiento de distintos algoritmos de clasificación mediante el diseño experimental apropiado y el análisis comparado de resultados. Además, también deberá extraer conclusiones a partir del conocimiento aprendido mediante estos algoritmos para comprender las relaciones entre las variables (también llamadas características o predictores) que favorecen una determinada clase. El trabajo se realizará sobre la plataforma KNIME (`http://www.knime.org`), pudiéndose emplear nodos adicionales de extensiones tales como Weka, JFreeChart o JavaScript Views.

2. Resultados obtenidos

En este apartado mostraré y comentaré los distintos algoritmos utilizados para esta práctica. También comentaré los resultados mostrando tanto tablas como imágenes.

En todas las imágenes de KNIME se podrá ver como al final de la predicción hay otros dos nodos, simplemente lo hice para filtrar las columnas que me interesan que en este caso eran la columna de la clase real, la columna de las predicciones y su correspondiente probabilidad de pertenecer a una clase u otra. Estos datos son utilizados para el cálculo de las métricas.

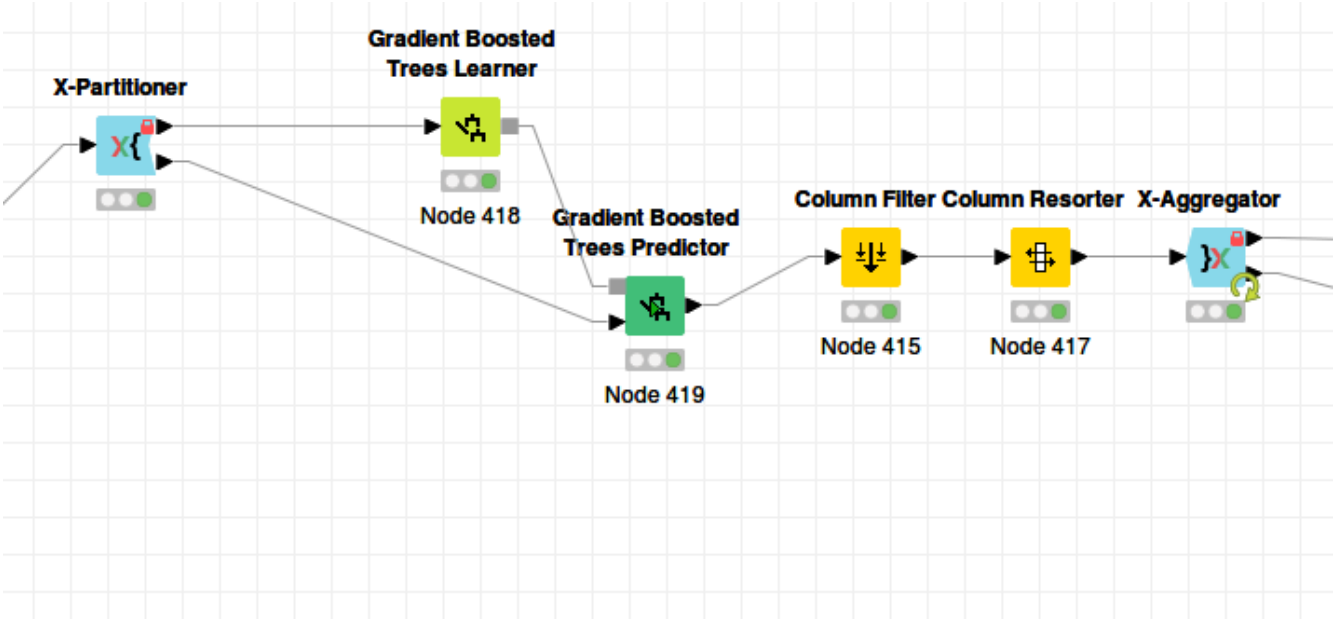
Todas las métricas se han calculado poniendo como clase principal la clase *"popular"*.

2.1. Gradient Boosted Trees

Este algoritmo usa arboles de regresion muy poco profundos y una forma especial de boosting para construir el conjunto de arboles. Los arboles de decision para este tipo de problemas dan buenos resultados ya que permite crear reglas de separación entre características que a su vez son interpretables para una persona (dependiendo del tamaño).

Los arboles de decisiones permiten trabajar con variables tanto numéricas como nominales, por lo tanto para este modelo no reconvertimos los datos. También permite que existan datos perdidos, por lo que por el momento dejaremos también esa parte sin tocarla.

Imagen KNIME de Boosted



Como resultado, se obtiene la siguiente tabla:

Tabla 1: Resultados obtenidos por el algoritmo Gradient Boosted Trees

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
Boosted	0,096	0,977	0,716	0,163	0,306	0,779

Tabla 2: Confusion Matrix para el algoritmo Gradient Boosted Trees

	no_popular	popular
no_popular	30013	705
popular	8070	856

Respecto a la complejidad del algoritmo, no he podido obtener la complejidad exacta del algoritmo utilizando el nodo *Gradient Boosted Tree Learner* por lo que pondré la configuración del mismo:

- Límite del número de niveles: 4
- Número de modelos: 100
- Ratio de aprendizaje: 0.1
- Muestreo de datos: NO
- Muestreo de atributos: NO
- Seleccion de atributos: Usa el mismo conjunto de atributos para cada árbol.

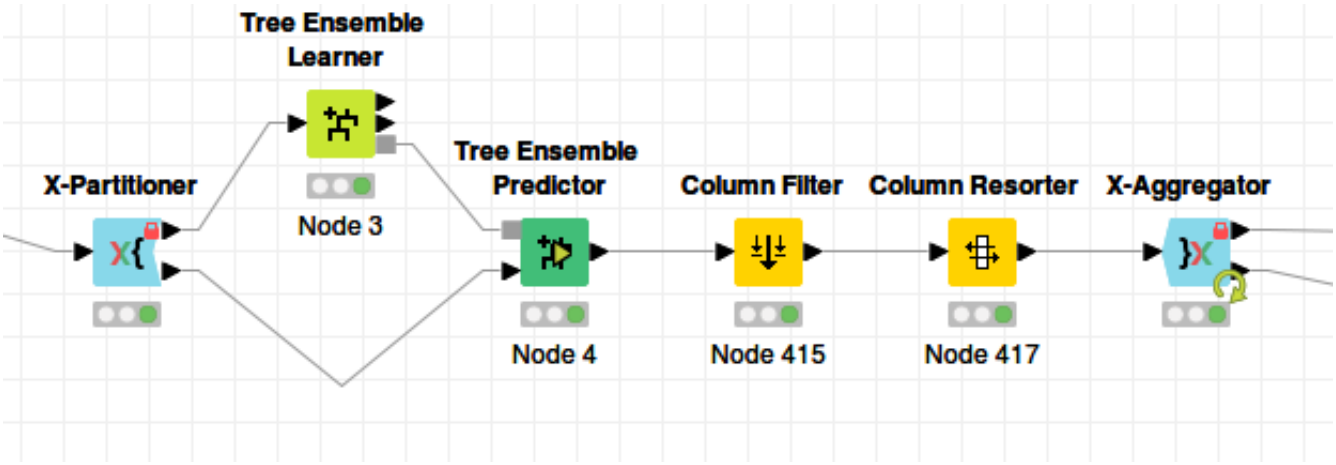
Todos los comentarios se harán en el siguiente apartado, comparando los distintos algoritmos utilizados.

2.2. Random Forest

Este algoritmo trabaja de una forma similar al anterior, ya que está compuesto por un conjunto de clasificadores simples para finalmente en un estudio de la clasificación de cada uno de ellos, devuelve la predicción final. Por lo que en este caso tenemos otro árbol de decisiones que como he comentado actúan muy bien en estos problemas.

Como este algoritmo puede utilizarse tanto con datos nulos y con con valores de texto, no tenemos que realizar ningun cambio.

Imagen KNIME de Random Forest



Del cual se obtiene como resultado:

Tabla 3: Resultados obtenidos por el algoritmo Random Forest

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
Random Forest	0,07	0,982	0,699	0,124	0,262	0,777

Tabla 4: Confusion Matrix para el algoritmo Random Forest

	no_popular	popular
no_popular	30180	538
popular	8303	623

Respecto a la complejidad del algoritmo, los valores utilizados han sido:

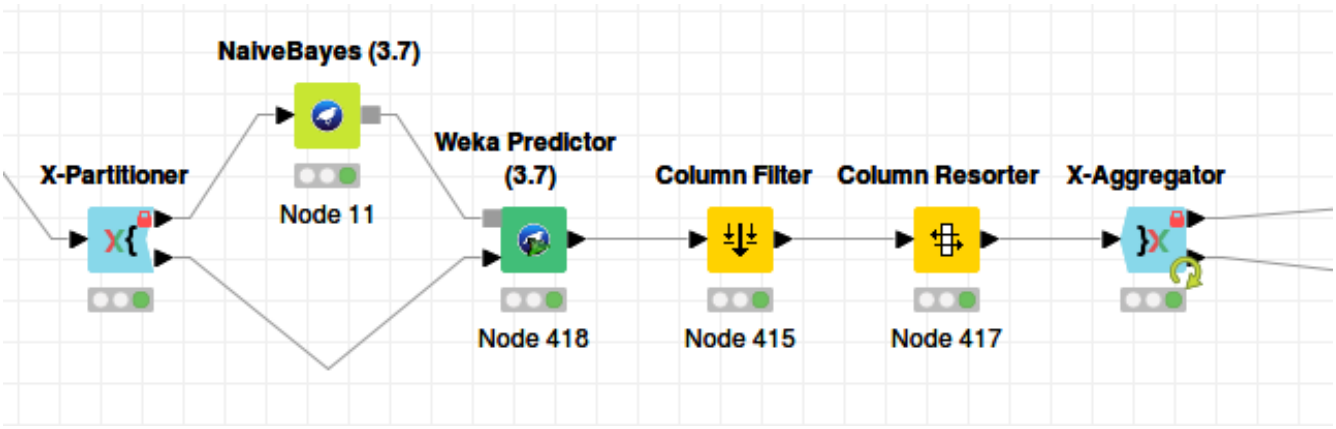
- Mínimo número de niveles: 81
- Máximo número de niveles: 167
- Número medio de niveles: 109.3
- Número de modelos: 100
- Mínimo numero de nodos: 7787
- Máximo número de nodos: 8191
- Número medio de nodos: 7974.66
- Muestreo de datos: Se selecciona una muestra del mismo tamaño de la muestra original pero escogiendo las instancias estratificadamente y con reemplazo.
- Muestreo de atributos: Se crea una muestra usando la raiz cuadrada.
- Seleccion de atributos: Usa distinto conjunto de atributos para cada árbol.

2.3. Naive Bayes

A diferencia de los anteriore, este clasificador es un algoritmo probabilístico, basado en el teorema de Bayes, de ahí su nombre. Al tener datos (mayoritariamente) numéricos, he pensado que este algoritmo puede funcionar correctamente y darle más diversificación al estilo de clasificadores que he escogido.

No obstante, este algoritmo permite variables no numéricas y también valores nulos, por lo que por ahora no tocaremos los datos.

Imagen KNIME de Naive Bayes



Del cual obtengo la siguiente tabla:

Tabla 5: Resultados obtenidos por el algoritmo Naive Bayes

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
Naive Bayes	0,101	0,944	0,61	0,155	0,308	0,754

Tabla 6: Confusion Matrix para el algoritmo Naive Bayes

	no_popular	popular
no_popular	28992	1726
popular	8028	898

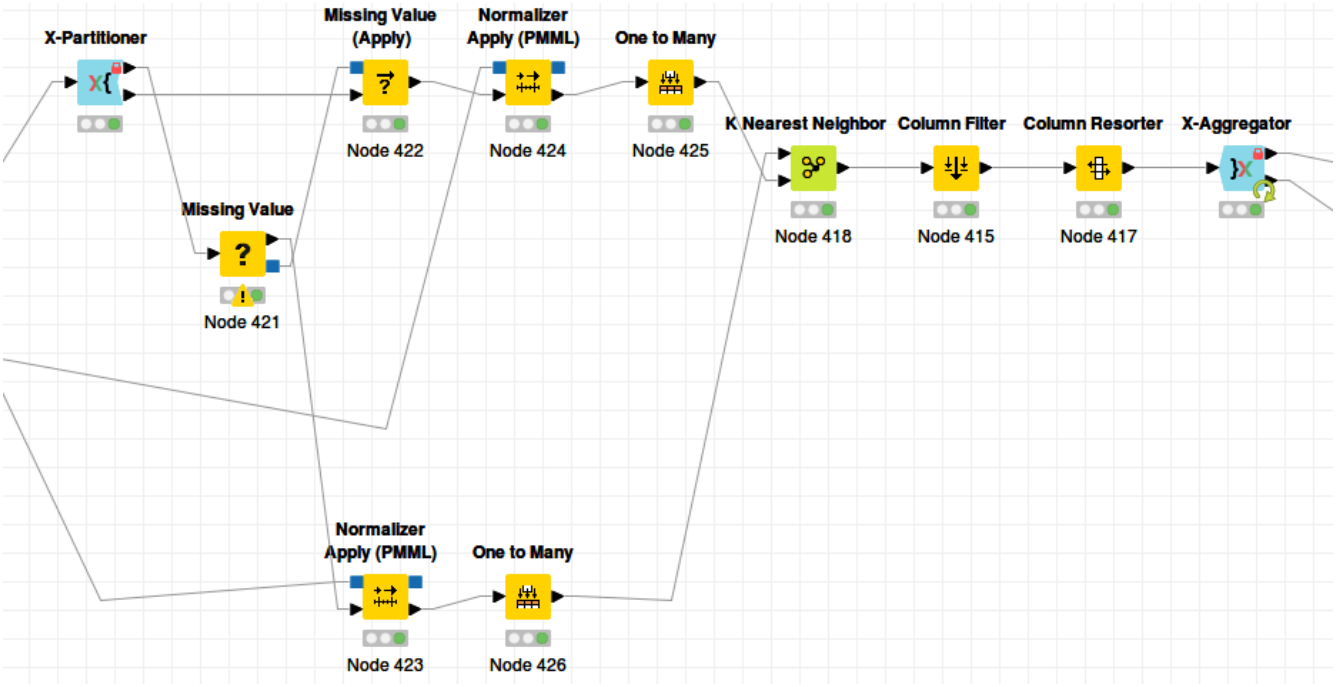
2.4. KNN

Este algoritmo es muy conocido y muy utilizado debido a su simplez, como en este caso tenemos un conjunto de datos donde la mayoría de los mismos son datos numéricos, el *KNN* puede resultarnos un buen clasificador ya que utiliza las distancias entre vecinos para clasificar a una clase. Claramente este algoritmo depende mucho de la relacion que haya entre vecinos y sus clasificaciones, no obstante he decido incluirlo en la eleccion de mis algoritmos para comprobar su rendimiento.

Para este caso, si que realizaremos una paqueña transformación en los datos para que el algoritmo funcione correctamente. He realizado una transformación de los campos de texto a numéricos y también he realizado una imputación simple en los datos, la imputación consiste en eliminar las

instancias que tengan valores nulos y también he normalizado los datos. Posteriormente se probarán otros métodos mas adecuados.

Imagen KNIME de KNN



Dando los siguientes resultados:

Tabla 7: Resultados obtenidos por el algoritmo KNN

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
KNN	0,173	0,914	0,598	0,235	0,398	0,748

Tabla 8: Confusion Matrix para el algoritmo KNN

	no_popular	popular
no_popular	21527	2019
popular	5613	1173

Respecto a la complejidad tenemos:

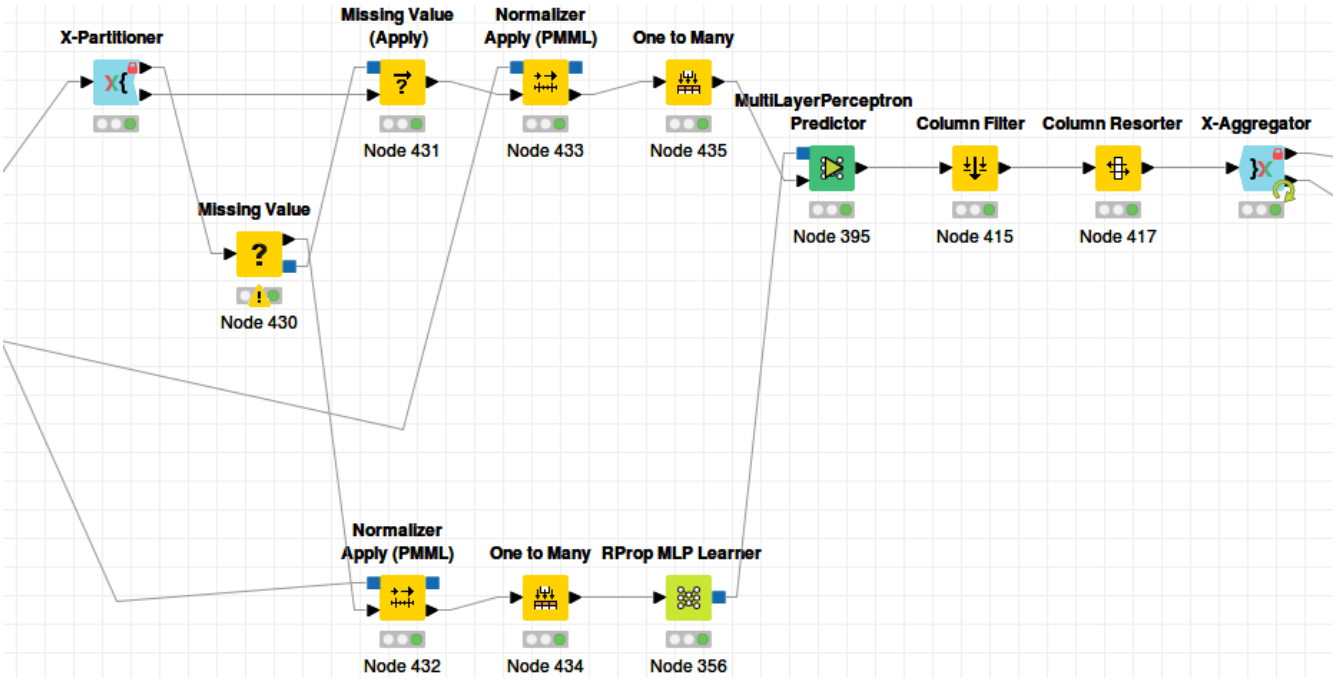
- Número de vecinos: 5

2.5. Redes Neuronales (ANN)

Este algoritmo lo he elegido por la calidad del mismo, aunque sabemos que las redes neuranles tienen una abstracción grande ya que no podemos saber lo que realizan en su procesamiento, son un algoritmo muy potente para predicción, por lo que este ha sido el principal motivo de la elección del mismo, no obstante también aumenta la diversificación de tipos de algoritmos que he seleccionado para comprobar el funcionamiento de ellos.

Estos algoritmos necesitan que los valores sean todos numéricos, que no hayan valores nulos y que los valores estén normalizados, por lo que este es el proceso que aplico a los datos para poder realizar este algoritmo. La imputación que realizo, como en el caso anterior es eliminar las instancias que tienen valores nulos ya que es la más simple, posteriormente se probarán otros métodos más recomendables.

Imagen KNIME de ANN



Dando los siguientes resultados:

Tabla 9: Resultados obtenidos por el algoritmo ANN

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
ANN	0,087	0,979	0,7	0,15	0,292	0,779

Tabla 10: Confusion Matrix para el algoritmo ANN

	no_popular	popular
no_popular	23045	501
popular	6194	592

Respecto a la complejidad, tenemos que:

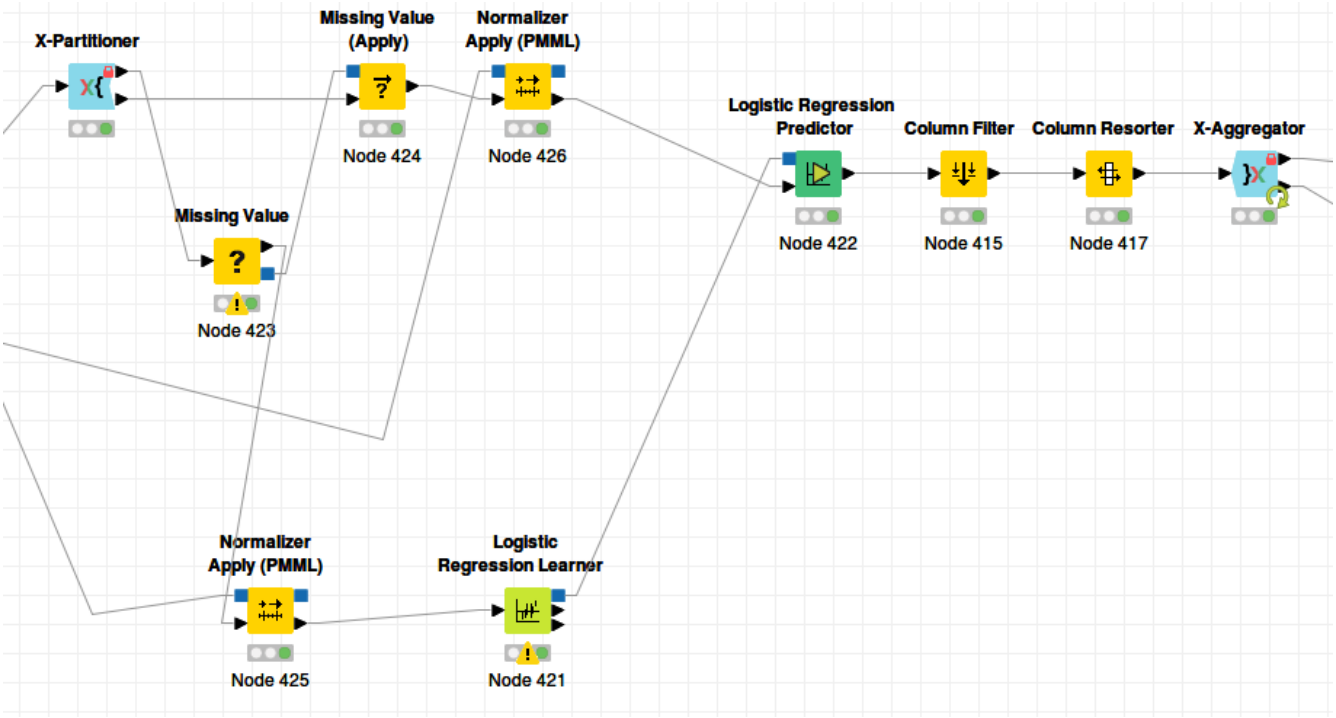
- Número máximo de iteraciones: 100
- Número de capas ocultas: 1
- Número de neuronas por capa: 10

2.6. Regresión Logística

En este caso utilizo otro algoritmo basado en cáclculos numéricos que nos devuelve la probabilidad de pertenecer a una clase dependiendo de otras variables.

Para este algorimo, los datos deben estar normalizados, no tener valores nulos y que todos los datos sean numéricos, aunque esta última característica no es obligatoria. Como en los casos anteriores, la imputación será eliminar las instancias y la normalización sera una normalización entre 0-1 como todos los apartados anteriores.

Imagen KNIME de Regresion Logistica



Dando los siguientes resultados:

Tabla 11: Resultados obtenidos por el algoritmo Regresion Logistica

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
Regresion Logistica	0,061	0,982	0,683	0,108	0,244	0,776

Tabla 12: Confusion Matrix para el algoritmo Regresion Logistica

	no_popular	popular
no_popular	23128	418
popular	6373	413

Respecto a la complejidad, tenemos que:

- Número máximo de épocas: 200
- Epslon: $1x10^{-5}$
- Tamaño del paso: 0.1
- Regularización: Uniforme

3. Análisis de los resultados

En este apartado me encargaré de discutir los resultados anteriormente obtenidos, mostrando gráficas y simplemente comentado los resultados.

Tabla resumen:

Tabla 13: Tabla resumen de métricas para todos los algoritmos

	TPR	**TNR**	**Area Under Curve**	**F-measure**	**G-mean**	**Accuracy**
Random Forest	0,070	0,982	0,699	0,124	0,262	0,777
KNN	0,173	0,914	0,598	0,235	0,398	0,748
Naive Bayes	0,101	0,944	0,610	0,155	0,308	0,754
ANN	0,087	0,979	0,700	0,150	0,292	0,779
Boosted	0,096	0,977	0,716	0,163	0,306	0,779
Regresion Logistica	0,061	0,982	0,683	0,108	0,244	0,776

Todas las métricas que tenemos en la tabla son valorables, pero en mi caso, he decidido hacer incapié en *AUC* y en *F-measure* ya que con *AUC* tenemos un visión sobre cuantos elementos realmente estamos clasificando correctamente y con *F-measure* tenemos una vision de cómo de bien estamos clasificando la clase positiva que para nuestro problema es lo que buscamos realmente.

Podemos ver en la tabla que mas o menos todos los algoritmos están equilibrados entre sí para las métricas que estamos midiendo. Podemos comprobar como *KNN* está dando unos resultados algo peores que el resto, esto se debe básicamente a su funcionamiento ya que éste es mucho más simple que el de los demás algoritmos. Este resultado más bajo que el resto nos dice que en la distribución que tenemos de los datos no tenemos unos datos que por ser de la misma clase deban estar juntos ya que si los elementos de una misma clase estubiesen juntos (para un método de cálcul ode distancia entre elementos) serían vecinos y por lo tanto *KNN* obtendía un mejor resultado. Sin embargo consigue el mejor *F-measure* de todos, lo que nos dice que es capaz de clasificar la clase positiva mejor que cualquier otro algoritmo de los que tenemos. Estos tipos de métricas para el *KNN* no son del todo representativas, ya que no devuelve ningún tipo de probabilidad de pertenecer a una clase, ésta se calcula dependiendo del número de vecinos que haya de cada clase, por lo que no es un cálculo exacto y es más bien orientativo.

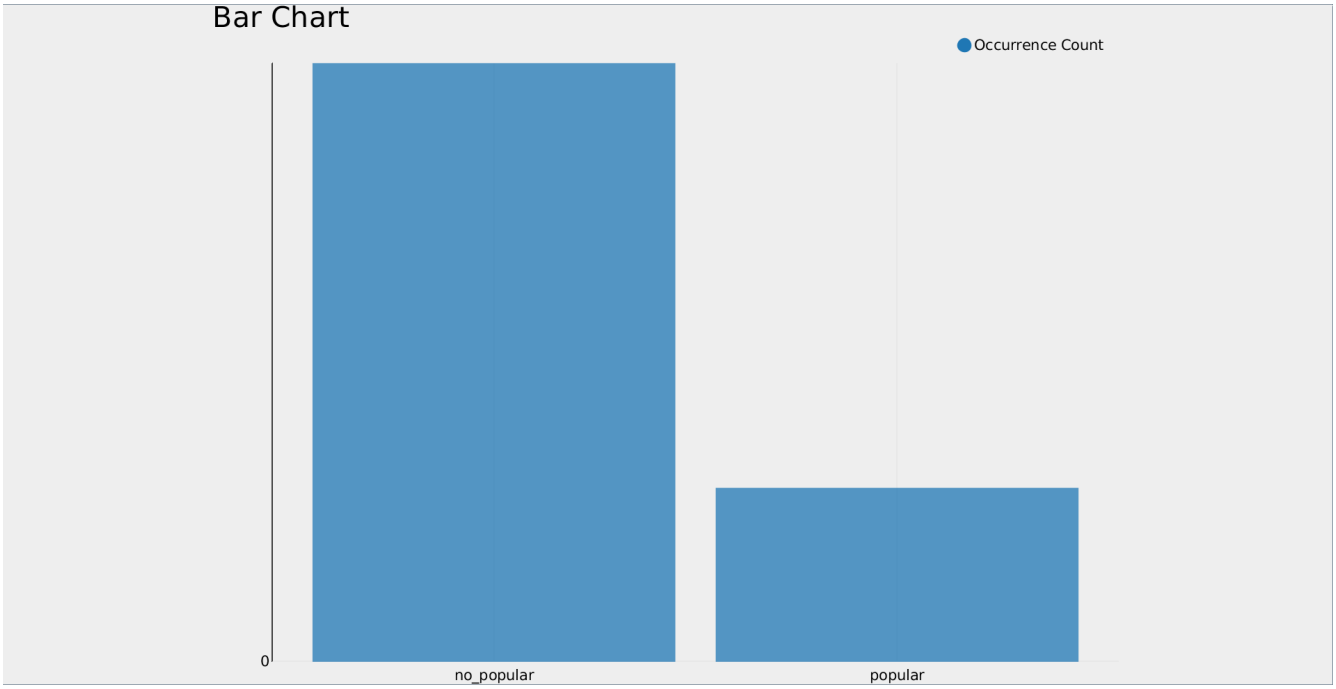
Por otra parte, podemos ver que los algoritmos que están en cabeza son el *ANN* y el *Boosted*, seguidos del *Random Forest* y por último el *Naive Bayes* y *Regresion Logistica*. Esto nos muestra que para este tipo de problema concreto, los algoritmos basados en árboles de decisión funcionan mejor que los algoritmos basados, en este caso, en probabilidades.

Sabiendo que estos algoritmos funcionan peor, por ejemplo con el caso de *Naive Bayes*, nos está diciendo que puede haber variables que no dependan de la clase o que hay dependencias entre variables (que viendo los datos claramente lo hay, ya que son publicaciones) ya que sabemos que en este punto es donde falla este algoritmo. Vemos que con *Regresión Logística* obtenemos unos resultados algo mejores siguiendo el mismo sentido que *Naive Bayes* aunque claramente utilizando otras propiedades para llegar a una conclusión algo mejor.

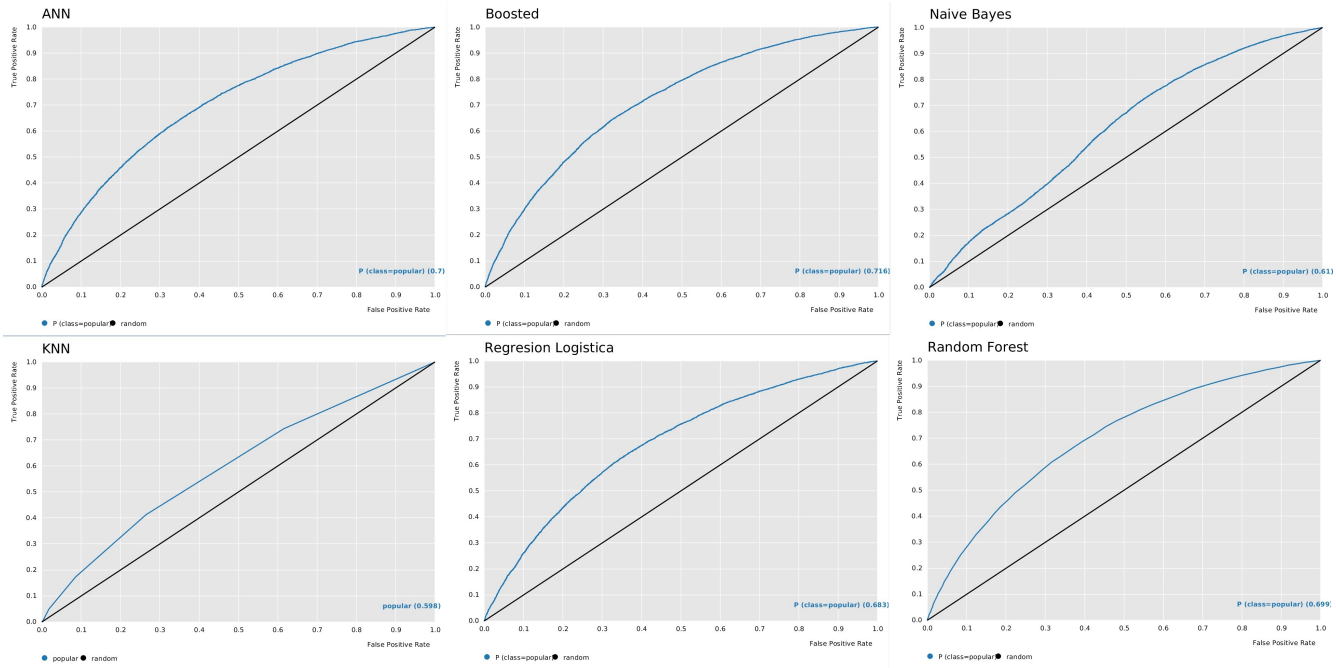
Como comentario final, tenemos que *ANN* y *Boosted* son los que mejor resultado nos están dando. Posiblemente *ANN* consiga unos de los mejores resultados debido a que al ser una red neuronal, sabemos que es robusta frente a ruido, outliers. etc... y como los datos que tenemos no han sido procesados, es posible que este algoritmo consiga mejor paliar este problema respecto a los demás algoritmos. No obstante, tenemos como cabeza de grupo al algoritmo *Boosted*, que como he comentado anteriormente, es un clasificador formado por un conjunto de clasificadores más smimples, utiliza Bagging para formar los conjuntos de datos y así entrenar a cada uno de los subclasificadores. Como las redes neuronales, los arboles de decisión tratan bien los datos con ruido, y al

obtener un buen resultado con este algoritmo, podemos que el problema se puede o por lo menos es mejor clasificarlo utilizando regiones rectangulares para dividir el dominio. Aunque tengamos los mejores valores en AUC para estos algoritmos, tenemos un valor bajo de *F-measure* que es una de las métricas que buscamos subir. Intentaremos en apartados posteriores mejorar este parámetro.

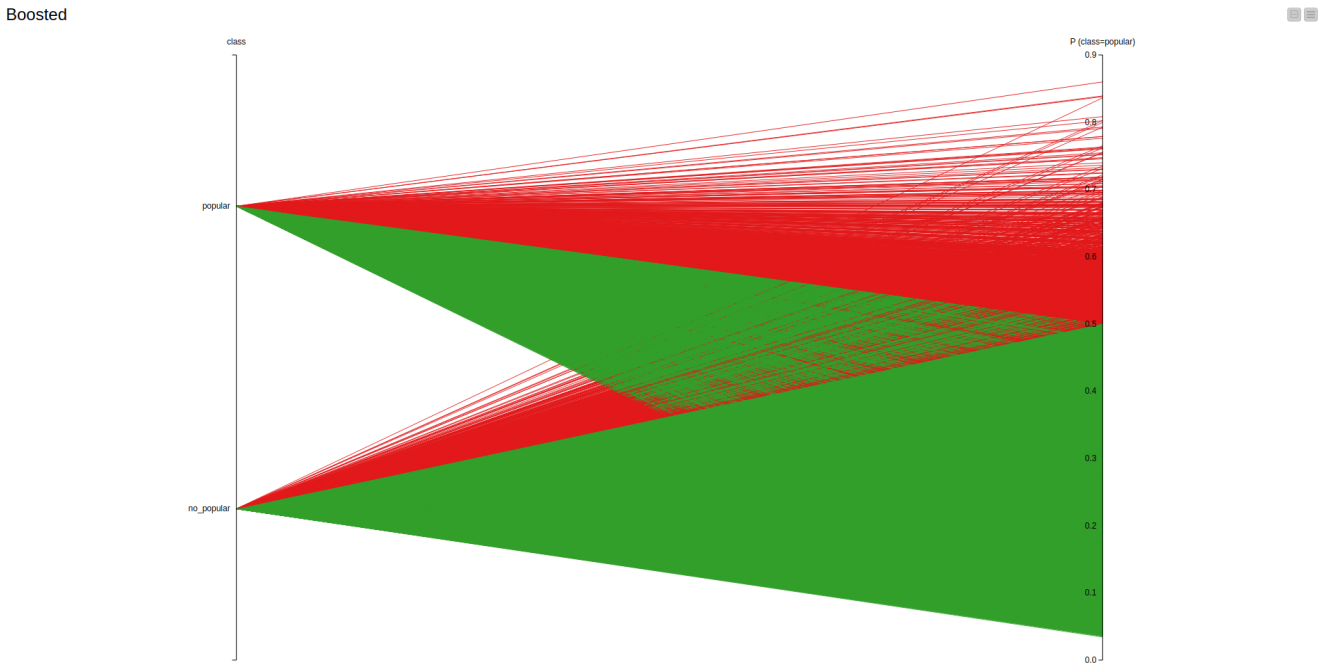
En cualquier caso, vemos como todos los algoritmos hacen un muy mal equilibrio entre tasa de acierto para la clase positiva y tasa de acierto para la clase negativa. Esto se debe a que tenemos un número de datos muy descompensado, ya que tenemos muchos más datos de la clase negativa que de la clase positiva y esto hace que los propios algoritmos no puedan clasificar con exactitud esta clase. Podemos ver en la siguiente gráfica la dedscompensación entre clases.



Por otra parte mostraré una gráfica con los resultados de las curvas ROC de cada uno de los algoritmos.



Por último, para mostrar la diferencia de clasificación respecto a la métrica *F-measure* que nos dice si estamos clasificando más o menos elementos de la clase positiva, mostraré en siguiente gráfico donde podemos ver como *Boosted* clasifica *(Verde = "no_popular" ;; Rojo = "popular").*



4. Configuración de algoritmos

En este apartado se realizará la modificación de los parámetros para cada uno de los algoritmos y la discusión de los resultados y diferencias entre ellos utilizando las mismas métricas vistas en apartados anteriores.

4.1. Gradient Boosted Trees

Para este algoritmo he decidido hacer dos modificaciones, primero, una modificación general, para abordar el problema de el sobreajustem por lo que he modificado los siguientes parámetros.

- Numero de modelos: 150
- Muestra de datos: 70 % sin reemplazamiento
- Muestra de atributos: Si, usando raiz cuadrada.
- Seleccion de atributos: Usar diferentes conjuntos de atributos para cada árbol.

Esta mejora la catalogo como **Mod1**

La siguiente modificación la realicé para que trabajase un poco mas y crease unos árboles un poco más extensos pero que nos puedan dar algo más de calidad, por lo que ahora he cambiado lo siguiente (manteniendo los cambios del modelo anterior):

- Limite de numero de niveles: NO

Este modelo lo he denominado **Mod2**

Como resultado, se obtiene la siguiente tabla:

Tabla 14: Resultados obtenidos por el algoritmo Gradient Boosted Trees

	TPR	**TNR**	**Area Under Curve**	**F-measure**	**G-mean**	**Accuracy**
Boosted	0,096	0,977	0,716	0,163	0,306	0,779
Mod1	0,107	0,975	0,717	0,179	0,323	0,779
Mod2	0,045	0,990	0,707	0,084	0,212	0,777

Podemos ver en los resultados como esta bastante complicado mejorar al algoritmo con los parámetros que tenía, no obstante, con la primera modificación donde utilizamos una porción de los datos para evitar el sobreajuste y la seleccion de características aleatorias, conseguimos una pequeña mejora en test y evitando sobreajuste por lo que esta mejora se puede considerar que aporta al algoritmo. Sin embargo, la segunda mejora, aunque le hemos dejado libertad para la profundidad de los árboles, no ha conseguido una mejora frente a los otros, incluso empeora los resultados que ya teníamos, por lo que podemos decir que esta mejora no aporta un buen resultado y que con una profuncidad fijada a 4 como se comentó en apartados anteriores, conseguimos una mejor solución.

4.2. Random Forest

Al igual que el anterior, he decidido hacerle dos cambios al algoritmo, donde para la primera modificación denominada **Mod1**, lo único que hago es cambiar el criterio de división por lo que tendŕiamos.

- Criterio de division: Gini Index

Por otra parte, la **Mod2** cambia aspectos más de la configuración de ejecución como puede ser el número de modelos, como algunos parámetros para evitar un poco el sobreajuste (se conserva el cambio anterior).

- Numero de modelos: 200
- Muestra de datos: 70 % sin remplazamiento estratificado.

Del cual se obtiene como resultado:

Tabla 15: Resultados obtenidos por el algoritmo Random Forest

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
Random Forest	0,070	0,982	0,699	0,124	0,262	0,777
Mod1	0,089	0,978	0,703	0,153	0,295	0,778
Mod2	0,073	0,982	0,709	0,128	0,268	0,777

Como vemos en los resultados, la mayor mejora la tenemos en la *Mod1* donde únicamente cambiando el criterio de división hemos conseguido mejorar un pequeño porcentaje los datos. El *Mod2* también mejora respecto al original, pero no respecto a la primera modificación, aunque vemos que por ejemplo en AUC obtiene un resutlado mejor que el resto a consta de perder un poco de *F-measure*, cosa que no nos interesa.

Respecto a la complejidad, se obtienen los siguiente modelos:

Tabla 16: Complejidad de los modelos para el algoritmo Random Forest

	Models	Min, Depth	Max, Depth	Average Depth	Min, Nodes	Max, Nodes	Average Nodes
Random Forest	100	86	143	111,38	7759	8211	7970,18
Mod1	100	32	46	37,11	11149	11529	11335,24
Mod2	200	30	51	35665	7785	8149	7953,31

4.3. Naive Bayes

Para Naive Bayes sólo he realizado una mejora, ya que no tiene tantos hyperparámetros configurables. Esta mejora **Mod1** consite únicamente en cambiar el parámetro que activa el uso de un kernel para estimar, por lo que los cambios realizados son:

- UseKernelEstimator: YES

Del cual obtengo la siguiente tabla:

Tabla 17: Resultados obtenidos por el algoritmo Naive Bayes

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
Naive Bayes	0,101	0,944	0,610	0,155	0,308	0,754
Mod1	0,010	0,997	0,625	0,019	0,098	0,775

Vemos como esta modificación consigue mejorarnos el porcentaje ded acierto en la clasificación, mejora tanto AUC como *Accuracy* pero a consta de bajar dŕasticamente *F-measure*, esto quiere decir que lo que hace ahora es acertar muchos más valores de la clase "no_popular", la clase negativa, a consta de fallar mucho en la clase positiva que es la que nos interesa a nosotros, por ello, aunque este algoritmo mejore en las métricas comentadas, por empeorar tanto en una que realmente nos conviene subir, daré esta nueva configuración como no valorable ya que desde mi punto de vista no es una mejora.

4.4. KNN

En este caso he realizado 3 cambios, uno para valorar la diferencia entre usar unos pesos para la clasificación valorando más los vecinos cercanos que los lejanos, y por otra parte en número de vecinos. Por lo que esto me ha dado los siguientes modelos.

Este primer modelo que sigue con la configuración del modelo original mas el siguiente cambio. Este modelo lo he denominado **Mod1**

- Pesos para los vecinos dependiendo de la distancia: SI

Para el siguiente modelo, **Mod2**, lo que he hecho es sobre el anterior, cambiar el número de vecinos.

- Número de vecinos: 3

Para el ultimo modelo, **Mod3**, lo que he hecho es como el anterior, cambiar el número de vecinos.

- Número de vecinos: 7

Dando los siguientes resultados:

Tabla 18: Resultados obtenidos por el algoritmo KNN

	TPR	**TNR**	**Area Under Curve**	**F-measure**	**G-mean**	**Accuracy**
KNN	0,173	0,914	0,598	0,235	0,398	0,748
Mod1	0,173	0,913	0,601	0,235	0,398	0,748
Mod2	0,215	0,879	0,584	0,263	0,435	0,731
Mod3	0,149	0,933	0,611	0,216	0,373	0,757

Vemos como el hecho de utilizar pesos, ya mejora en el caso del *Mod1* respecto al original, si aumentamos el número de vecinos 5 *(Mod2)*, obtenemos un mejor resultado para *F-measure* cosa que es un cambio importante ya que es lo que buscamos, a cambio, perdemos un porcentaje de AUC y *Acurracy*, pero vamos a tener que ir acostumbrandonos a ello, porque si queremos mejorar *F-measure* clasificando mejor la clase positiva, perderemos fuerza en los valores de AUC y *Acurracy* generalmente.

4.5. Redes Neuronales (ANN)

Para este algoritmo he realizado también dos cambios, comprobando qué valores se ajustan mejor para este problema, finalmente por una parte tenemos **Mod1** que aumenta tanto en numero de niveles de la red como el número de neuronas que hay en cada una de los niveles.

- Numero de capas ocultas: 5
- Numero de neuronas por capa: 15

Para la segunda modificacion **Mod2**, se mantienen los cambios anteriores, pero aumentamos tambien el numero de iteraciones.

- Numero de iteraciones: 300

Dando los siguientes resultados:

Tabla 19: Resultados obtenidos por el algoritmo ANN

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
ANN	0,087	0,979	0,700	0,150	0,292	0,779
Mod1	0,120	0,965	0,685	0,194	0,341	0,776
Mod2	0,187	0,932	0,671	0,262	0,417	0,765

Como he ido comentando en los apartados anteriores, tenemos que sacrificar un poco en los valores de *AUC* y *Accuracy* si queremos mejorar *F-measure*. En este caso se ve claramente como al red permite mejorar la clasificación de la clase positiva y con ello mejorar el *F-measure*, cosa que determino que es una mejora aunque otros valores bajen. La *Mod2* es el que mejor resultado obtiene ya que tiene la misma configuracion que *Mod1* pero le permitimos más iteraciones para conseguir un mejor resultado.

4.6. Regresión Logística

Por último, para este caso, he realizado una única mejora, basada en las iteraciones/épocas que realiza el algoritmo ya que por defecto tienen un número bastante bajo y no converge, aunque tampoco le puedo dar un número muy alto para que converja porque si no se excedería demasiado en tiempo.

Esta modificación la he denominado **Mod1**

- Numero de epocas: 1500

Dando los siguientes resultados:

Tabla 20: Resultados obtenidos por el algoritmo Regresion Logistica

	TPR	TNR	Area Under Curve	F-measure	G-mean	Accuracy
Regresion Logistica	0,061	0,982	0,683	0,108	0,244	0,776
Mod1	0,064	0,981	0,683	0,113	0,250	0,776

Podemos ver como el algoritmo a sido capaz de mejorar *F-measure* muy levemente pero sin perder *AUC* ni *Acurracy*. Una prueba podría ser seguir aumentandole las épocas para ver los resultados que van saliendo, siempre y cuando se disponga del tiempo necesario que tardará en resolverlo.

5. Procesado de datos

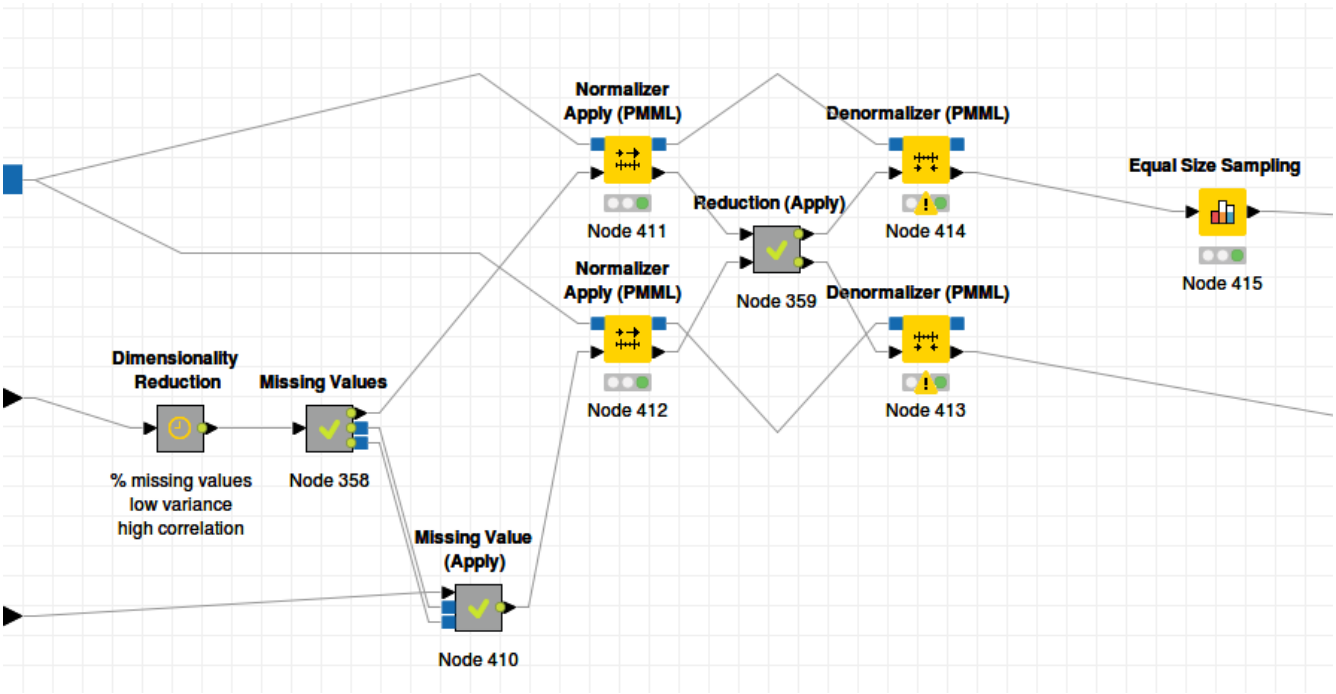
En este apartado, al haber tanto algoritmos he decidido mostrar sólo los 2 que mejor resultado he obtenido hasta el momento. Para el preprocesado, he realizado un preprocesado común para todos, uno que creo que es básico para la mejora de la predicción. Como aclaración, lo que busco en este apartado es mejorar el valor de *F-measure* subiendo el ratio de acierto sobre la clase positiva, esto nos lleva a una pérdida de algunas otras métricas pero que prefiero perder a consta de mejorar esta.

Antes de nada comentar que implementé un nodo para realizar la eliminación del ruido utilizando el algoritmo IPF, sin embargo, como esta eliminación ha de hacerse dentro de cada una de las

iteraciones del cross-validation, llevaba un tiempo excesivo realizando tantas veces para todos los algoritmos, por lo que decidí quitarlo. Sólo quería comentarlo para indicar que en nodo se encuentra dentro del proyecto de KNIME y que en todos los casos mejora los resultados respecto a los algoritmos sin este proceso.

Para el preprocesado he seguido los siguientes pasos (algoritmos con campos no numericos):

- En primer lugar, he realizado una reduccion de dimensionalidad muy leve, basada en el porcentaje de valores perdidos que tiene una variable. Esto es que si una variable concreta tiene más del 90 % de valores perdidos, esta variable será eliminada.
- En segundo lugar trato los datos perdidos que queden, en mi caso realizo una interpolacion linal para rellenar los datos y seguidamente si han quedado datos sin rellenar cojo la media del valor esa variable. Para el caso de las variables no numéricas, se coje el valor de la más común.
- Despues de esto, normalizo los datos para seguidamente calcular la matriz de correlacion de las variables y hacer una reduccion de dimensionalidad eliminando las variables que tengan un 80 % de correlacion. Seguidamente elimino las variables que tengan una varianza menor de 0,05. Una vez hecho esto denormalizo los datos para los algormitmos que no necesiten normalizado y en caso de necesitarlo los dejo como estan.
- Por último, he decidio hacer un equilibrado de clases ya que éste es el principal problema por el que no conseguimos hacer una buena predicción de la clase positiva, este método mejorá ese valor en contra de otros.



Para el preprocesado he seguido los siguientes pasos (algoritmos únicamente numericos):

- En primer lugar, he realizado una reduccion de dimensionalidad muy leve, basada en el porcentaje de valores perdidos que tiene una variable. Esto es que si una variable concreta tiene más del 90 % de valores perdidos, esta variable será eliminada.

- En segundo lugar trato los datos perdidos que queden, en mi caso realizo una interpolacion linal para rellenar los datos y seguidamente si han quedado datos sin rellenar cojo la media del valor esa variable.
- Despues de esto, normalizo los datos para seguidamente calcular la matriz de correlacion de las variables y hacer una reduccion de dimensionalidad eliminando las variables que tengan un 80 % de correlacion. Seguidamente elimino las variables que tengan una varianza menor de 0,05. Una vez hecho esto denormalizo los datos para los algormitmos que no necesiten normalizado y en caso de necesitarlo los dejo como estan.
- Despues, realizo un equilibrado de clases ya que éste es el principal problema por el que no conseguimos hacer una buena predicción de la clase positiva, este método mejorá ese valor en contra de otros.
- por último, paso las variables no numéricas a variables numéricas para que el algoritmo pueda trabajar con ellas.

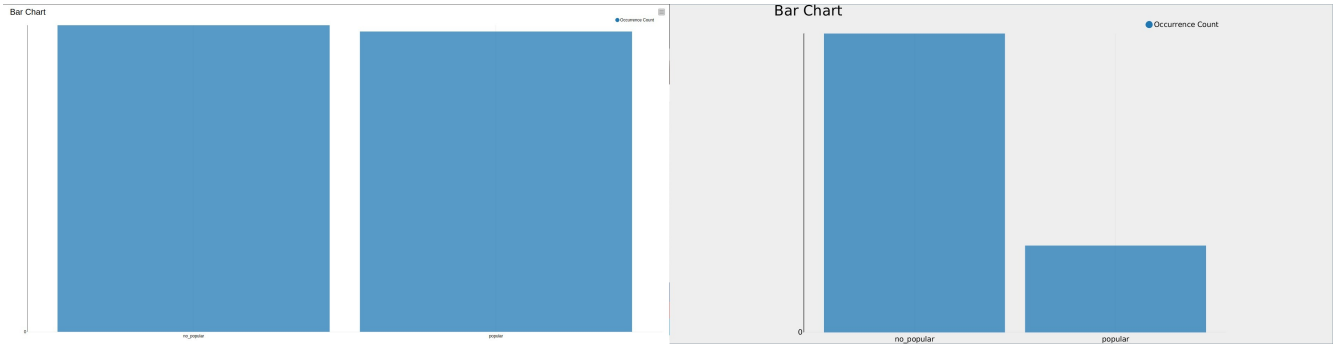
Despues de realizar este preprocesado para los algoritmos *Boosted*, *ANN* y *KNN* se obtienen los siguientes resultados

Tabla 21: Resultados obtenidos con Preprocesado

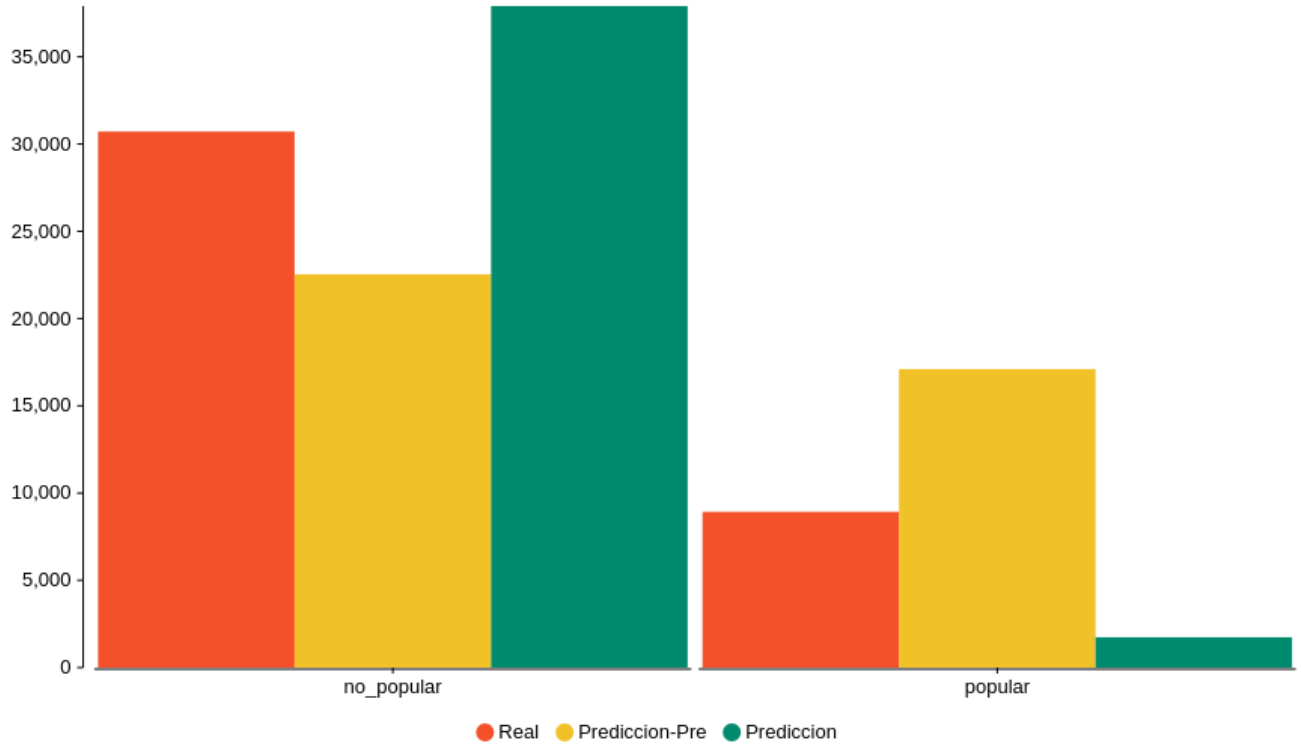
	TPR	**TNR**	**Area Under Curve**	**F-measure**	**G-mean**	**Accuracy**
Boosted	0,639	0,629	0,678	0,438	0,634	0,631
KNN	0,571	0,597	0,607	0,386	0,584	0,591
ANN	0,617	0,638	0,668	0,431	0,627	0,633

Podemos ver como claramente mejora *F-measure* que es lo que buscaba principalmente aunque hayamos empeorado un poco algunas otras métricas. No obstante enseñaré distintas gráficas comparativas con partes anteriores para diferenciar más visualmente los resultados.

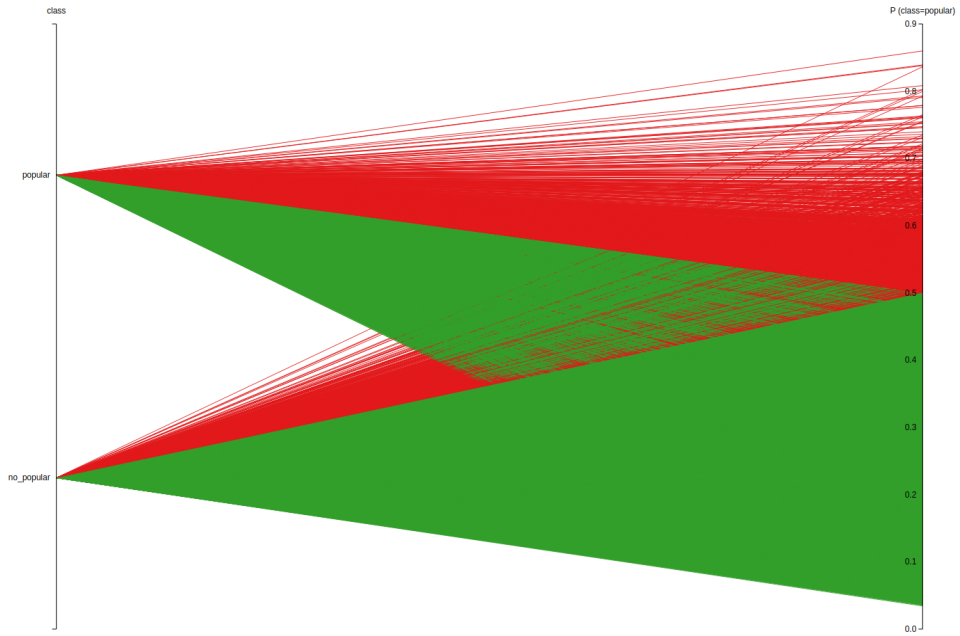
Distrubucion de clases en el conjunto de datos:

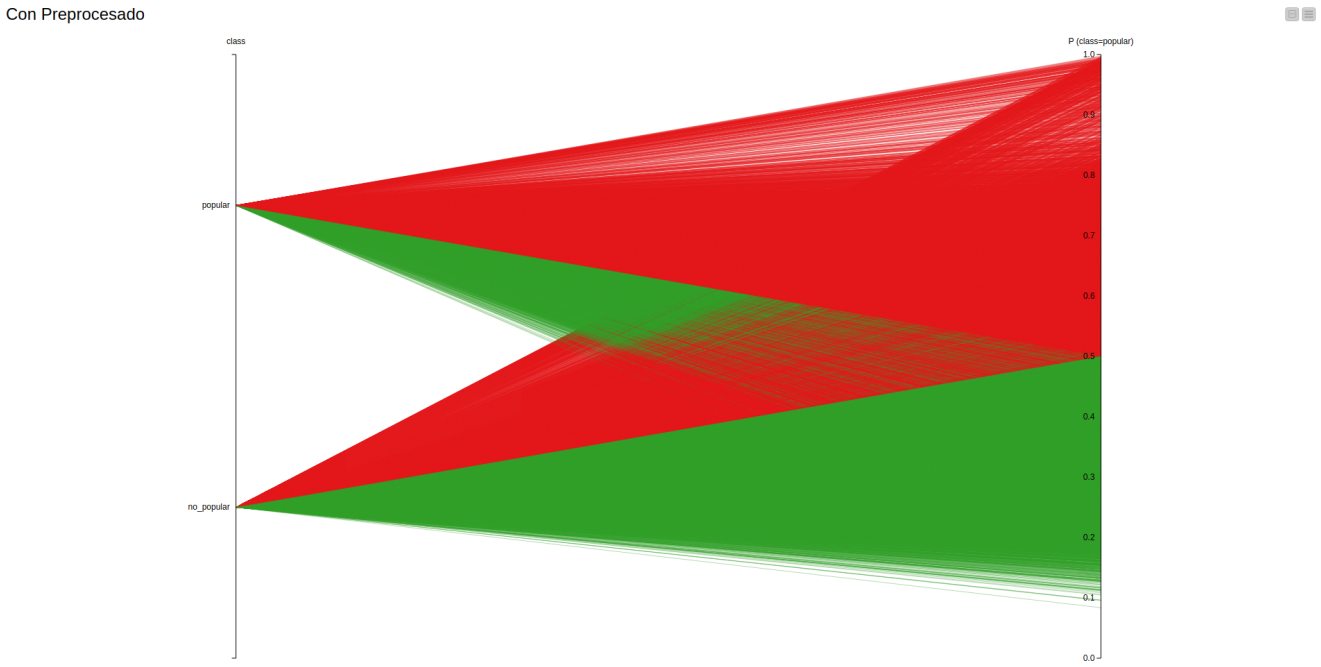


Elementos que acierta antes y despues del preprocesado.


35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
no_popular
popular
Real
Prediccion-Pre
Prediccion

Boosted

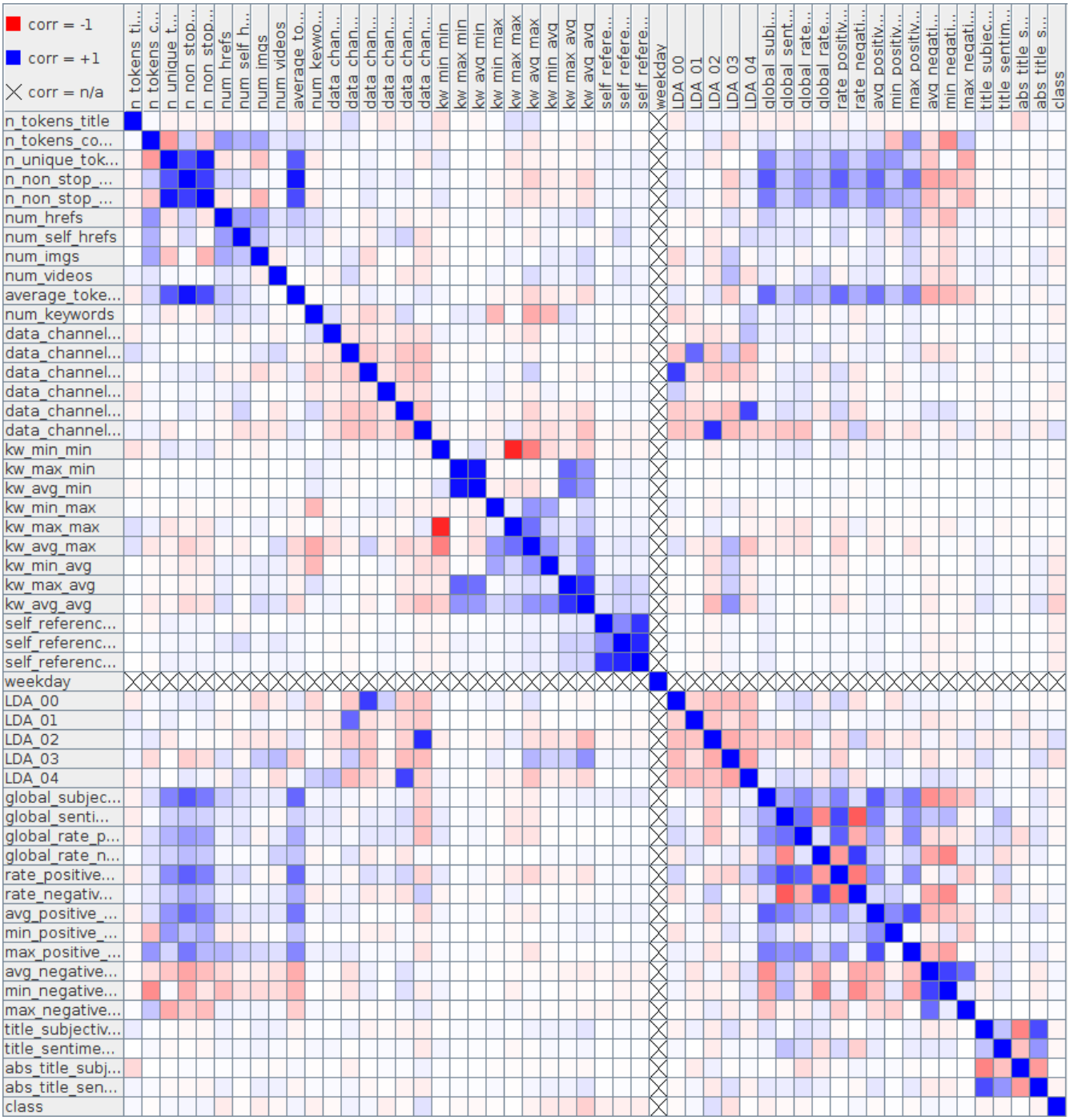

class
P (class=popular)
popular
no_popular
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

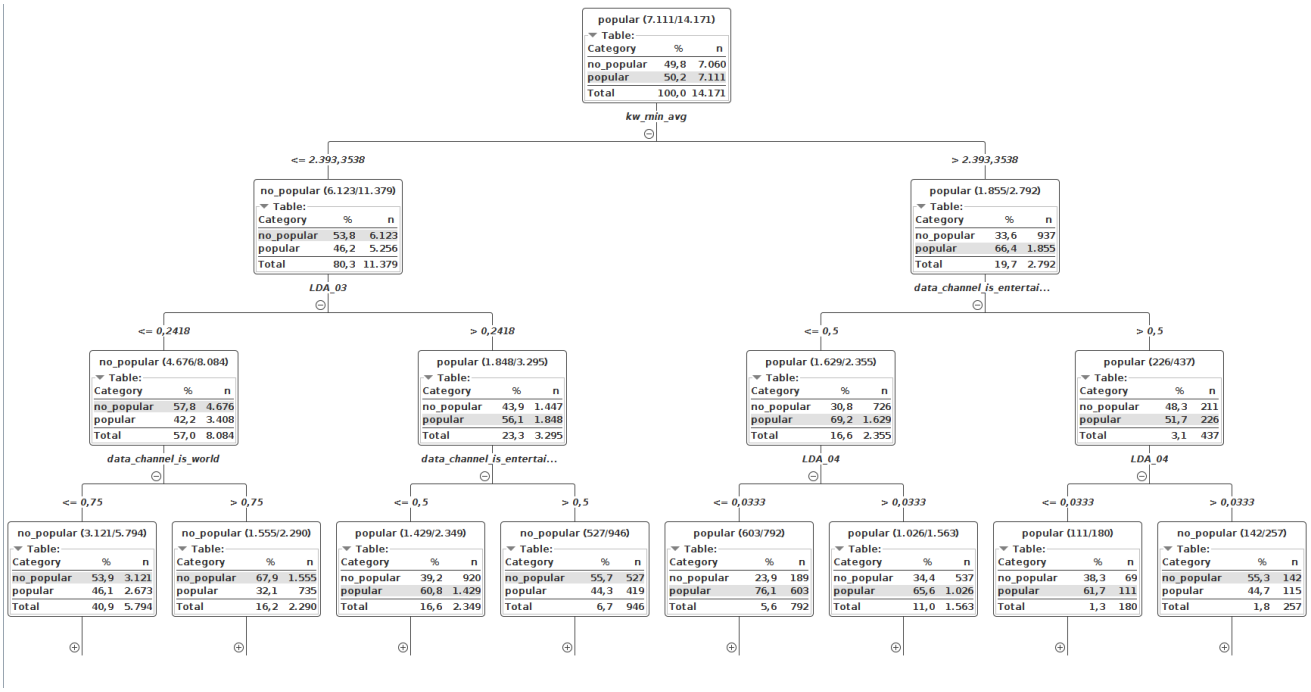


Podemos ver en las gráficas como despues del preprocesado es capaz de predecir mucho mejor la clase positiva respecto a anteriormente cuando no realizabamos preprocesado, por lo que podemos concluir que este preprocesado hace su funcion correctamente.

6. Interpretación de los resultados

En esta sección mostraré la matriz de correlación de los datos para comentar un poco lo importante que puede ser una varaible, después he construido un arbol de decisión algo más simple para poder extraer el árbol de decisiones y ver las variables que él cataloga como mas importantes para la clasificación.


corr = -1
corr = +1
corr = n/a
n_tokens_title
n_tokens_co...
n_unique_tok...
n_non_stop_...
n_non_stop_...
num_hrefs
num_self_hrefs
num_imgs
num_videos
average_toke...
num_keywords
data_channel...
data_channel...
data_channel...
data_channel...
data_channel...
data_channel...
kw_min_min
kw_max_min
kw_avg_min
kw_min_max
kw_max_max
kw_avg_max
kw_min_avg
kw_max_avg
kw_avg_avg
self_referenc...
self_referenc...
self_referenc...
weekday
LDA_00
LDA_01
LDA_02
LDA_03
LDA_04
global_subjec...
global_senti...
global_rate_p...
global_rate_n...
rate_positive...
rate_negativ...
avg_positive_...
min_positive_...
max_positive_...
avg_negative...
min_negative...
max_negative...
title_subjectiv...
title_sentime...
abs_title_subj...
abs_title_sen...
class



En la primera imagen, donde vemos la matriz de correlación podemos ver que algunos de los campos se van a eliminar en el proprocesado, estos son los campos que salen como azules oscuros o como rojos fuertes. Los campos que estan blancos, son atributos que no tienen correlacion con nadie y que por lo tanto los necesitamos para poder predecir las clases con mas exactitud.

Por otra parte, en la imagen del árbol vemos que la carácterística que mejor separa las clases principalmente es *k2_min_avg*, ya que dependiendo de este valor depende una gran diferencia a pertenecer a una clase o utra. Después se va finjando en otros como *LDA_3* para un caso y en *data_chanel_is_entertai...* en el otro caso. Sucesivamente podemos ver los diferentes atriburos que va mirando para comprobar si realmente pertence a una clase u otra y con esto podríamos intuir o construir unas reglas que nos dijeran la clase a la que pertenece.

Referencias

[1] Web asignatura, sci2s.ugr.es, `https://sci2s.ugr.es/graduateCourses/in`, Accedido el 4 de noviembre de 2018.

[2] Web de KNIME, knime.com, `https://www.knime.com/` Accedido el 4 de noviembre de 2018

[3] Web para diversos plots, plotdb.com, `https://plotdb.com/` Accedido el 4 de noviembre de 2018