Inteligencia de Negocio

Miguel Morales Castillo

miguemc@correo.ugr.es
Grupo Lunes

Noviembre 2018

Índice

1. Introducción

En esta práctica aprenderemos a hacer análisis predictivo empresarial mediante algoritmos de aprendizaje supervisado, que realizan clasificación. En concreto, vamos a intentar, utilizando dichos algoritmos, predecir cuando una noticia es popular, las clases posibles son popular y no popular. Trabajaremos con el conjunto de datos OnlineNewsPopularity. En él, se almacenan 39644 ejemplos, y para cada ejemplo se almacenan 52 características, tanto numéricas como categóricas. Además, existen algunos datos perdidos. Esto nos obligaría a hacer un procesamiento previo de los datos, que nos permita adecuarlos a los algoritmos que vamos a utilizar a posteriori para intentar extraer información.
En cuanto a los modelos que vamos a utilizar, aplicaremos árboles de decisión, Naive Bayes, redes neuronales, random forest, kNN, y gradient boosted tree. Todos los experimentos se realizarán usando la técnica 5-CV, es decir, validación cruzada de 5 particiones. Consiste en dividir el conjunto de datos en cinco partes, y repetir el experimento 5 veces, usando cada vez una de las partes como conjunto de test, y las cuatro restantes como conjunto de entrenamiento.

La división de la experimentación será la siguiente. En la primera parte de la práctica, aplicaremos los modelos haciendo el preprocesamiento mínimo necesario para que dichos algoritmos funcionen. En la siguiente, atendiendo a los resultados obtenidos en la primera parte, haremos un preprocesamiento mas profundo para tratar de mejorar los resultados, teniendo una versión con los parámetros por defecto y otra con los parámetros modificados.

Toda la experimentación de la práctica se realizará utilizando la plataforma de análisis de datos KNIME. Es una plataforma desarrollada por la Universidad de Constanza, Alemania, que permite el desarrollo de modelos de aprendizaje automático y minería de datos, utilizando para ello un entorno de desarrollo visual basado en módulos.

Comenzamos la práctica mostrando los resultados obtenidos por los modelos implementados con el menor preprocesamiento posible.

2. Resultados Obtenidos

Vamos a ver los resultados obtenidos por los seis modelos seleccionados para el conjunto de datos dado. Como ya hemos dicho antes, los modelos seleccionados son el árbol de decisión C4.5, Naive Bayes, MLP , random forest, un clasificador kNN, y gradient boosted tree. Comenzamos comentando la experimentación con boost gradient tree, así como los resultados obtenidos por el mismo.

2.1. Boost Gradient Tree

Como ya sabemos, este modelo se basa en el cálculo de modelos simples, que se utilizan para calcular una función que minimice la probabilidad de que se produzca un error al intentar clasificar un nuevo dato. Un ejemplo de uso muy típico es combinar esta metodología de trabajo con árboles de decisión, y es precisamente el método de trabajo que implementaremos.

Veamos los resultados que obtenemos con este modelo en nuestro conjunto de datos:

	Predicción	
Predicción	*no popular*	*popular*
no popular	30495	223
popular	8859	67

Cuadro 1: Matriz de confusión del modelo *gradient boosted*

Ahora, obtenemos un porcentaje de acierto del 77,1 %, con una sensibilidad de 0,008 y una especificidad de 0,993. En la siguiente sección haremos un análisis comparativo de los resultados obtenidos por todos los modelos, para intentar averiguar cuáles son los modelos que mejor funcionan con nuestro conjunto de datos, así como para intentar mejorar los resultados de los algoritmos que peor se comportan.

2.2. Naive Bayes

Veamos los resultados obtenidos por este modelo. Como sabemos, este modelo utiliza el teorema de Bayes, asumida la independencia de las características almacenadas, para calcular la probabilidad de que un elemento pertenezca a una determinada clase dados los valores de los atributos del mismo. Esta asunción de independencia, que en muchos casos no es cierta, es necesaria para poder aplicar el teorema de Bayes, pero puede dar lugar a malos resultados. No obstante, como veremos a continuación, los resultados obtenidos no son excesivamente malos.

Mostramos la matriz de confusión para este modelo:

	Clase	
Predicción	*no popular*	*popular*
no popular	30717	1
popular	8926	0

Cuadro 2: Matriz de confusión del modelo *Naive Bayes*

Ahora, el porcentaje de acierto conseguido por el modelo es del 77,5 %, con una sensibilidad del 0, y una especificidad del 1. Que salga un porcentaje de acierto tan alto a pesar de clasificar todas las instancias como negativas se debe a que la base de datos esta muy mal balanceada, ya que como podemos ver tiene 30718 instancias de la clase no popular y solo 8926 de la clase popular.
Veamos algunas de las tablas que nos genera este modelo para hacer las predicciones. Distinguimos dos tipos de tablas, una para los atributos numéricos, en los que se supone que los mismos siguen una distribución normal, y otra para los atributos nominales, en los que se calcula la probabilidad utilizando la regla de Laplace.

Mostramos las tablas mencionadas, una correspondiente a un atributo numérico y otro a un atributo nominal:

	no popular	*popular*
elementos:	24575	7141
media:	9025.51593	14505.33763
desv. típica:	37020.64353	50653.18935
ratio:	77 %	23 %

Cuadro 3: Distribución gaussiana para el atributo self_reference_max_shares

clase	Lunes	Martes	Miercoles	Jueves	Viernes
no popular	4091	4673	4792	4611	3534
popular	1162	1213	1290	1166	1023
Ratio:	17 %	19 %	19 %	18 %	14 %

Cuadro 4: Distribución por día de la semana

Usando estas tablas, se puede calcular la probabilidad de que un nuevo objeto a clasificar pertenezca a una determinada clase, atendiendo a sus características.

2.3. Árbol de decisión C4.5

Estudiamos ahora el árbol de decisión. Como ya sabemos, este modelo se construye seleccionando características que definen los ejemplos de nuestro conjunto de datos, y creando nuevos nodos en el árbol atendiendo a los posibles valores que tome esa característica. De esta forma, podemos clasificar una nueva instancia comenzando por la raíz del árbol, y bajando niveles dependiendo de los valores que la instancia tome para la característica en cuestión. Repetimos este proceso hasta alcanzar una hoja, que estará etiquetada con la clase correspondiente que se asocia a la instancia a clasificar.

El algoritmo de construcción del árbol trata de seleccionar en cada caso los atributos que más información aportan, usando para ello distintos índices que se nos permite seleccionar. Nosotros trabajaremos con el índice de Gini, que nos permite medir la probabilidad de etiquetar incorrectamente un elemento aleatoriamente si atendemos a la distribución de etiquetas en un conjunto de datos. El uso de esta medida nos permitirá, a posteriori, interpretar qué atributos nos aportan más información sobre nuestros elementos del conjunto, viendo qué características se seleccionan en los primeros nodos de nuestro conjunto. Esto lo haremos inspeccionando el árbol de decisión que se construya, ya que KNIME nos permite explorar el árbol una vez construido.

Se muestran a continuación los resultados obtenidos al entrenar este modelo con la técnica de 5-CV. Comenzamos mostrando la matriz de confusión:

	Clase	
Predicción	*no popular*	*popular*
no popular	24000	6718
popular	6263	2663

Cuadro 5: Matriz de confusión del árbol de decisión

Obtenemos que el porcentaje de acierto de este modelo es del 67,3 %, con una sensibilidad de 0,298 y una especificidad de 0,7813. Vamos a continuación a explorar el árbol de decisión construido, para intentar obtener información relevante sobre las variables más explicativas de nuestro modelo. No

adjuntaremos la vista completa del árbol, ya que es bastante extenso y no sería fácil interpretarlo, así que sólo se adjuntan algunos de los niveles:

Podemos observar que la característica más influyente es self_references_min_shares. Es por esto que es la primera característica que se selecciona. Otra característica muy relevante es data_channel_is_world, que nos aparece como segunda característica seleccionada. Es lógico que esta característica sea muy relevante, ya que cuanto mayor sea el alcance del canal donde se difunde la noticia, parece evidente que mayor será la probabilidad de que dicha noticia consiga ser popular.

Si ahondamos un poco más en el árbol, otra característica que aparece repetida en varias ocasiones es el número de referencias que tiene la noticia. Usualmente, las noticias mas compartidas en internet, también conocidas como "virales"son las que se vuelven mas populares.

2.4. Random Forest

Veamos el resultado obtenido por *random forest*. Este modelo se basa en la construcción de varios árboles de decisión, que clasifican los nuevos objetos de forma independiente, y después se somete el resultado final a votación. De nuevo, este modelo no necesita de preprocesamiento, ya que los árboles de decisión no los necesitaban. Los parámetros que se utilizan son 100 árboles, con un límite de 10 niveles de profundidad. Veamos los resultados que se obtienen:

	Clase	
Predicción	*no popular*	*popular*
no popular	30718	0
popular	8926	0

Cuadro 6: Matriz de confusión del modelo *random forest*

Ahora, tenemos un porcentaje de acierto del 77,5 %, con una sensibilidad del 0, y una especificidad de 1. Vemos que el modelo actua de forma similar al Naive Bayes. Por contra, el modelo pasa a ser más difícilmente interpretable, ya que ahora tenemos un conjunto mucho mayor de árboles que estudiar que en el C4.5, lo que puede ser largo y tedioso. No obstante, KNIME nos permite visualizar los mismos y estudiarlos individualmente, en caso de que necesitemos de esa

información. Con un vistazo general, sin desarrollar nada más que el primer nodo del árbol, podemos observar que para muchos de los árboles el nodo más importante es el asociado a la variable $global_sentiment. También aparece en muchos árboles el nodo asociado a la$

Veamos los resultados obtenidos al entrenar una red neuronal sobre este conjunto de datos. Para este modelo hemos tenido que realizar un pequeño preprocesamiento, ya que las redes neuronales necesitan que todas las variables sean numéricas. Esto nos ha obligado a convertir las variables categóricas a variables numéricas. Además, hemos hecho una normalización de los datos, para evitar que ciertas categorías fuesen más influyentes unas que otras.
Además, después de realizar el particionado, se han tratado los valores perdidos. Este modelo no funciona bien si hay valores perdidos en los datos, por lo que se decide tratar los mismos. A los valores perdidos se les ha asignado el valor más repetido.

En cuanto a la configuración de la red neuronal, utilizamos los parámetros por defecto que asigna KNIME, esto es, una red neuronal, con 3 capas ocultas, y 10 nodos por capa. El número máximo de iteraciones de entrenamiento que se realizan son 100.

Los resultados obtenidos son los siguientes:

	Clase	
Predicción	*no popular*	*popular*
no popular	30717	1
popular	8926	0

Cuadro 7: Matriz de confusión de la red neuronal

Tenemos ahora un porcentaje de acierto del 77,5 %*, con una sensibilidad del* 0 *y una especificidad del* 1*. Vemos que el funcionamiento de la red neuronal actúa como el resto de algoritmos clasificando todas las instancias como la mayoritaria debido al desbalanceo . Una de las dificultades que presenta es que no podemos interpretar los resultados obtenidos en función de las variables de entrada, ya que este tipo de modelos no nos muestra una estructura interpretable de su funcionamiento. Es por esto por lo que no podemos saber qué características son más relevantes en este modelo, lo que dificulta su refinamiento en apartados posteriores.*

2.5. K-Nearest Neighbors

Veamos los resultados que obtenemos usando un modelo basado en kNN. Como ya sabemos, la idea de este modelo consiste en calcular la distancia del nuevo elemento a clasificar y todos los datos de nuestro conjunto, seleccionar las k distancias menores, y quedarnos con la clase que más veces se repita de entre las k seleccionadas. En el caso de la configuración por defecto, se toma $k = 3$*, y no se pondera la importancia de las clases en función de la distancia calculada, es decir, los k individuos más cercanos se consideran igualmente relevantes, independientemente de la distancia a la que se encuentren. Existe la posibilidad de dar más importancia a los elementos más cercanos al individuo a clasificar, opción que discutiremos más adelante.*

Al igual que nos pasó con la red neuronal, necesitamos eliminar las variables categóricas, convirtiéndolas a variables enteras. Además, necesitamos normalizar los datos, para que todas las características sean igualmente significativas (en caso de que los valores no estuvieran normalizados, las características con mayor diferencia entre su valor mínimo y máximo serían mucho más influyentes en la clasificación que las que tienen un rango de valores reducido). Finalmente, hemos tenido que gestionar los valores perdidos. Hemos usado el mismo procedimiento que usamos con la red neuronal. Veamos a continuación la matriz de confusión para este modelo:

	Clase	
Predicción	*no popular*	*popular*
no popular	26698	4020
popular	7373	1553

Cuadro 8: Matriz de confusión del clasificador kNN

Conseguimos ahora una sensibilidad de 0,174 *y una especificidad de* 0,869*, con un porcentaje de acierto del* 71,3 %*. Observamos que este clasificador nos arroja resultados bastante pobres en comparación con muchos de los anteriores aunque no clasifica todo como la clase mayoritaria, por lo que puede ser un buen candidato para tratar de mejorar, ajustando sus parámetros y con un preprocesamiento más exhaustivo.*

3. Análisis de Resultados

Vamos a comparar ahora los resultados obtenidos por los algoritmos anteriores. Esto nos servirá para ver qué modelos funcionan mejor con el conjunto de datos a tratar. Una vez veamos los resultados, podremos intentar mejorarlos.

Se muestra a continuación una tabla resumen con la información de los resultados extraída por KNIME. Se considera que la clase positiva es ser popular:

Modelo	V positivos	F positivos	V negativos	F negativos	Accuracy	Sensibilidad	Especificidad	F1-score
Gradient Boosted	67	223	30495	8859	0.771	0.008	0.993	0.015
Naive Bayes	0	1	30717	8926	0.775	0	1	0
Árbol de decisión	2663	6718	24000	6263	0.673	0.298	0.781	0.291
Random Forest	0	0	30718	8926	0.775	0	1	0
Red neuronal	0	1	30717	8926	0.775	0	1	0
kNN	1553	4020	26698	7373	0.713	0.174	0.869	0.214

Cuadro 9: Tabla resumen de los resultados obtenidos por los modelos

Vamos a comentar los resultados obtenidos. Lo primero que podemos observar claramente es el desbalanceo que existe entre las dos clases. Tenemos sólamente 8926 datos de la clase positiva, mientras que de la clase negativa tenemos 30718 ejemplos. Esto dificulta el aprendizaje, especialmente el reconocimiento de los verdaderos positivos.

En cuanto al funcionamiento de los algoritmos, como podemos observar en la tabla, el modelo que nos consigue un mayor número de verdaderos positivos es el modelo de KNN. Viene seguido por gradient boosted. Si observamos el número de verdaderos negativos, tenemos que los algoritmos que mejor se comportan son random forest y la red neuronal, seguidos por gradient boosted. Después aparecen con un comportamiento similar el KNN y Naive Bayes, y con peores resultados el árbol de decisión.

Centrémonos ahora en la comparativa de los modelos que utilizan árboles de decisión. El que tiene peor comportamiento es el árbol de decisión simple, que tiene los valores de sensitividad y especificidad más bajos de los tres. Está seguido por random forest en cuanto a sensibilidad, y finalmente el que consigue los mejores resultados es gradient boosted. En cuanto a especificidad, ambos algoritmos arrojan resultados muy similares, por lo que podemos decir que gradient boosted es el mejor algoritmo de los tres, ya que, aunque consigue menos accuracy, permite detectar verdaderos positivos, no clasifica todas las instancias como la clase negativa.

Un motivo que puede haber influido también en este problema es la forma en la que hemos tratado las características nominales, que son además las que más aparecen en nuestros ejemplos. El preprocesamiento que les hemos dado ha sido simplemente convertirlas a variables enteras, asignando, de forma un tanto arbitraria (KNIME hace esta asignación por orden alfabético), un número natural a cada uno de los valores posibles que podía tomar la característica. Este artificio nos puede llevar a malos resultados.

Una vez hechas estas reflexiones, en el apartado siguiente comentaremos cómo hemos intentado resolver estos problemas de comportamiento de dichos algoritmos, así como los nuevos resultados obtenidos.

4. Configuración de Algoritmos

Pasamos a la configuración de los parámetros de los algoritmos. En primer lugar, trabajaremos sobre el árbol de decisión.

4.1. Ajuste de parámetros del árbol de decisión

Comenzamos viendo las posibilidades de configuración de parámetros que nos ofrece KNIME para el árbol de decisión. Cuando entramos en la configuración del nodo, se nos ofrecen las siguientes opciones:

Figura 1: Opciones de configuración del árbol de decisión

Nosotros trabajaremos en concreto con la medida de la calidad de selección del nodo, que alternaremos entre el índice de Gini (por defecto) y el ratio de ganancia, y el método de poda, que puede tomar los valores "sin poda" y "poda MDL" (Minimum description length pruning), que a grandes rasgos consiste en eliminar las ramas del árbol que menos información aportan, es decir, aquellas que necesitan de muchos nodos, para explicar muy pocos ejemplos del conjunto de entrenamiento (la no eliminación de dichas ramas lleva a problemas de sobreajuste). Usaremos poda con ganancia. Los resultados que se obtienen son los siguientes:

Modelo	V positivos	F positivos	V negativos	F negativos	Accuracy	Sensibilidad	Espeficificidad	Valor F
Gini con poda	1367	2269	28449	7559	0.752	0.153	0.926	0.218

Cuadro 10: resultados para la configuración

Vamos a comentar los resultados obtenidos. En primer lugar, se observa claramente que la poda mejora significativamente los resultados, especialmente en la clase positiva. Esto indica que sobre esta clase se estaba produciendo cierto sobreaprendizaje. La poda nos ha servido para mejorar los resultados sobre dicha clase. Esto nos llevaría a escoger como modelo el que utiliza el indice Gini, y realizando poda MDL.

Pasamos ahora a ver los resultados obtenidos para Naive Bayes.

4.2. Ajuste de parámetros para Naive Bayes

En esta sección comentaremos los resultados obtenidos para Naive Bayes. Cuando entramos en la configuración de dicho modelo, los parámetros que podemos modificar son los siguientes:

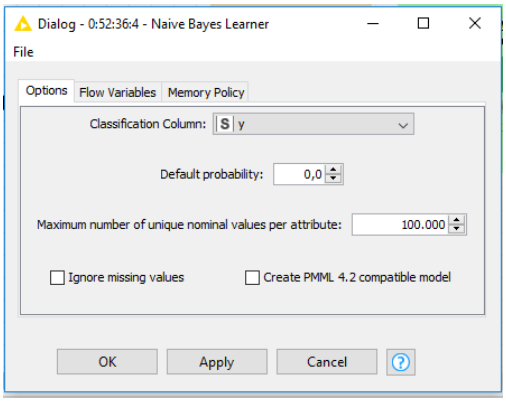


Figura 2: Opciones de configuración de *Naive Bayes*

En este caso, lo que se nos permite modificar es la probabilidad de que un determinado elemento pertenezca a la clase positiva a priori, así como el máximo número de valores que puede tomar un atributo nominal para ser considerado dentro del modelo.

El primer parámetro se utiliza para hacer correcciones en caso de clases desbalanceadas. No obstante, para asignar valores a dicha probabilidad, necesitamos de más información sobre el problema de la que tenemos, por lo que lo dejaremos siempre a 0. El otro, nos permite descartar para la clasificación las características nominales que tengan demasiados valores. Esto se hace porque si un determinado atributo tiene muchos valores posibles, la probabilidad de que un determinado objeto pertenezca a dicha clase es muy baja, lo que puede provocar que los resultados obtenidos tengan cierto ruido.

Trabajaremos modificando ese parámetro. Le daremos suficientemente alto, para que se incluyan todas las caracteristicas .

Veamos los resultados obtenidos:

Modelo	V positivos	F positivos	V negativos	F negativos	Precisión	Sensibilidad	Especificidad	Valor F
100000 valores	3391	7509	23209	5535	0.671	0.38	0.756	0.342

Cuadro 11: resultados de la configuración

En este caso, podemos observar que la utilización de mas variables nos lleva a una mejoría en la sensibilidad del algoritmo, acompañada de una pérdida de especificidad. Aunque el accuracy sea peor podemos considerar que hemos mejorado los resultados ya que ahora somos capaces de obtener un mayor número de verdaderos positivos.

Vemos ahora el ajuste de parámetros para la red neuronal:

4.3. Ajuste de parámetros para la red neuronal

En el caso de la red neuronal, los parámetros que se pueden configurar son los siguientes:

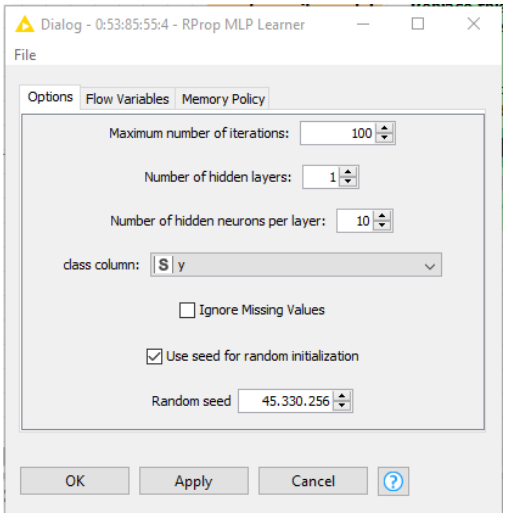


Figura 3: Opciones de configuración de la red neuronal

Como se observa en la imagen, podemos modificar, esencialmente, tres valores. Por un lado, el número de iteraciones de aprendizaje que se realizan (es decir, el número de veces que se hace backpropagation en el proceso de entrenamiento de la red neuronal). Por otro, el número de capas ocultas que componen la red neuronal, y finalmente el número de neuronas que componen cada capa. Se puede echar de menos la posibilidad de configurar cada capa individualmente, cambiando el número de neuronas que componen cada unidad por separado.

Vamos a desarrollar cinco modelos distintos. Complicaremos un poco mas el modelo añadiendo tantos nodos como características, y manteniendo el numero de iteraciones y capas ocultas capa.

Los resultados obtenidos en este caso son los que siguen:

Modelo	V positivos	F positivos	V negativos	F negativos	Precisión	Sensibilidad	Espeficificidad	Valor F
52 neuronas	830	944	29774	8096	0.772	0.093	0.969	0.6687

Cuadro 12: Resultados de la configuración

Podemos observar que ahora las diferencias entre las distintas configuraciones no es especialmente significativa. Los resultados obtenidos al complicar el modelo son esencialmente iguales a los que obtenía el algoritmo por defecto, salvo por la ventaja de que ahora es capaz de clasificar mas verdaderos positivos.

Pasamos finalmente a ver los ajustes que se pueden realizar para el modelo kNN.

4.4. Ajuste de parámetros para kNN

Para este modelo, los posibles parámetros a configurar son los siguientes:

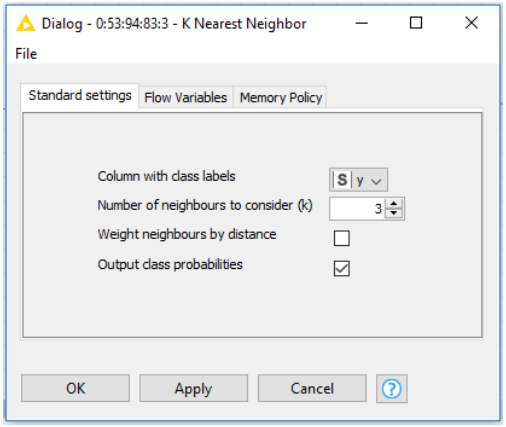


Figura 4: Opciones de configuración para kNN

Principalmente, podemos modificar dos parámetros. El número de vecinos que se tienen en cuenta, y si las distancias están o no ponderadas. Probaremos cómo se ve modificado el resultado ponderando las distancias. A continuación, se mostrará un caso particular de kNN, cuando $k = 1$, que consiste básicamente en asociar a cada dato nuevo a clasificar la clase del objeto más cercano de entre los que tenemos en el conjunto de test. En este caso, la ponderación respecto a la distancia no tiene sentido, por lo que no haremos dos ejecuciones distintas. Finalmente, se observará la influencia que tiene el aumento en el número de vecinos, tanto con los votos ponderados como con votación simple. Los resultados obtenidos son los siguientes:

Modelo	V positivos	F positivos	V negativos	F negativos	Accuracy	Sensibilidad	Especificidad	Valor F
5-NN + dist. ponderadas	1317	2953	27765	7690	0.734	0.148	0.904	0.2

Cuadro 13: resultados para las configuración

En este caso, los resultados muestran que el aumento de complejidad del modelo sí que se traduce en una mejora de los resultados obtenidos por el algoritmo.

La elección de la complejidad del modelo, en este caso, es similar a la que hemos tenido en apartados anteriores. De nuevo tenemos que ver si nos interesa tener un modelo más complicado, a cambio de mejores resultados, aunque el tiempo de cómputo sea mayor, o preferimos un modelo más simple, en el que los resultados sean ligeramente peores.

Una vez terminada la experimentación, pasamos a hacer describir el procesamiento realizado.

5. Procesado de datos

En esta sección vamos a realizar un preprocesado más inteligente de los datos, para mostrar la importancia que tiene dicha operación en los resultados finales obtenidos. Hemos realizado un preprocesamiento generico para todos los algoritmos, que consiste en discretizar las características nominales, tratar los valores perdidos y normalizar los datos, además de introducir una calculadora de dominios. A continuación, pasamos a explicar el preprocesado especial aplicado a algunos de los algoritmos.

5.1. Preprocesamiento para el árbol de decisión

Como hemos comentado en la introducción, este modelo funciona bastante mal cuando las clases están muy desbalanceadas. Es por esto por lo que vamos a intentar compensar este desbalanceo, ya que por ser además en la clase positiva, la sensibilidad del modelo es muy baja.

Existen dos enfoques distintos que podemos utilizar para tratar de compensar el problema que nos hemos encontrado. Por un lado, podemos muestrear el conjunto de entrenamiento de forma igualitaria, de forma que se desprecien datos de la clase mayoritaria. De esta forma, obtenemos un conjunto de datos balanceado, pero con muchos menos datos de entrenamiento. La ventaja de usar este método es que computacionalmente es muy eficiente, ya que consiste simplemente en filtrar la tabla de entrada por clase, seleccionar todos los datos que pertenecen a la clase minoritaria, y seleccionar aleatoriamente tantos datos de la clase mayoritaria como ejemplos de la clase minoritaria tenemos.

El otro enfoque consiste en crear, de forma artificial, ejemplos de datos de la clase minoritaria. En este enfoque, KNIME implementa un nodo que nos permite realizar este sobremuestreo. Utiliza la técnica SMOTE, que consiste en crear datos de forma artificial de la siguiente manera. Para crear un nuevo dato de la clase minoritaria, se escoge aleatoriamente un dato que pertenezca a dicha clase, se buscan los k vecinos más cercanos, se selecciona aleatoriamente uno de ellos, y los valores de los atributos del nuevo objeto se calculan haciendo una combinación convexa de los valores de los dos objetos seleccionados (esto es, para cada atributo, se genera aleatoriamente un valor $\alpha \in [0, 1]$, y se construye $v_{nuevo} = \alpha v_1 + (1 - \alpha)v_2$. Este proceso se repite hasta que se construyan un número de ejemplos ficticios predeterminado por el usuario. En el caso de KNIME, podemos especificar el coeficiente de sobremuestreo que se realiza, esto es, si llamamos k a dicho coeficiente, el cardinal del conjunto de datos de la clase minoritaria tras el procesamiento con SMOTE será $1 + k$ veces mayor que el cardinal del conjunto de datos minoritarios antes del procesamiento.

Es evidente que este procesamiento es computacionalmente mucho más costoso que el que hemos comentado anteriormente. La creación de datos ficticios requiere de la generación de múltiples va-

lores aleatorios, los cuales suelen requerir de mucho tiempo de cómputo. Además, la búsqueda de los vecinos más cercanos requiere del cálculo de distancias, que es una operación costosa. Además, para aplicar esta técnica, las variables categóricas pierden el sentido, ya que como hay que calcular distancias, necesitamos que todas las variables sean numéricas. Es importante en este punto decidir qué estrategia seguir, ya que la elección de una u otra puede provocar resultados no deseados, dependiendo del problema real que queramos resolver. La elección del undersampling, que es la primera técnica que hemos explicado, nos conduce a una reducción importante del tamaño de datos que tenemos para aprender, lo cual puede ser un desperdicio de información muy valiosa, ya que los datos de entrenamiento son la única fuente de información que tenemos para resolver nuestro problema. La elección del oversampling, segunda estrategia explicada, se traduce en un aumento del tiempo de cómputo muy significativo, además de en la pérdida de interpretabilidad del modelo. Un punto fuerte de los árboles de decisión es la interpretabilidad del modelo calculado, ya que la estructura del árbol es legible. La conversión de las variables categóricas a variables binarias dificulta en gran medida la interpretación de resultados, por lo que si el modelo se construye en vistas de ser interpretado por expertos en el problema a resolver, este enfoque queda prácticamente invalidado. Por tanto usaremos undersampling.

Mostramos a continuación los resultados obtenidos en ambos casos:

Modelo	V positivos	F positivos	V negativos	F negativos	Precisión	Sensibilidad	Especificidad	Valor F
Con undersampling	4935	13829	16889	3991	0.55	0.553	0.0.55	0.356

Cuadro 14: Comparativa tras el ajuste de clases

Con el undersampling, obtenemos un accuracy considerablemente peor, pero por contra obtenemos un balance muchisimo mejor entre sensibilidad y especificidad, por lo que para según que problemas puede ser una buena solución.

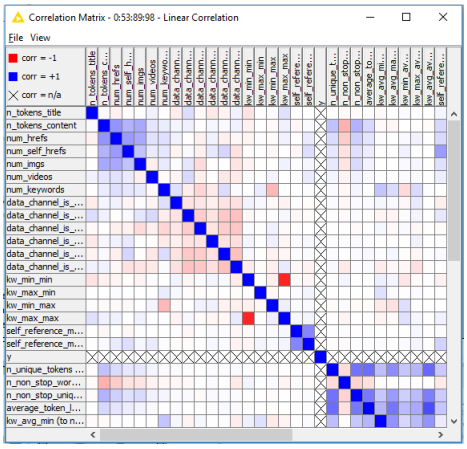
Pasamos a ver el preprocesamiento realizado para el modelo basado en Naive Bayes.

5.2. Preprocesamiento para *Naive Bayes*

Para este modelo, como comentamos en la introducción, vamos a intentar mejorar la sensibilidad eliminando las dependencias que puedan existir entre los atributos. Ya vimos anteriormente que este algoritmo suponía la independencia estadística entre las variables contempladas. No obstante, en ningún momento hemos comprobado si las variables que se contemplan en nuestro problema son realmente independientes. La dependencia entre ellas podría causar problemas a la hora de utilizar este modelo.

Para estudiar la dependencia o independencia de nuestros datos, vamos a utilizar un nodo implementado en KNIME llamado linear correlation. Dicho nodo nos permite calcular la dependencia lineal de nuestras variables dos a dos. Observando la tabla que se genera, podemos ver si existe correlación o no entre ellas. Se muestra a continuación la tabla que genera knime tras aplicar la correlación lineal:

Vamos a comentar la tabla. La salida que obtenemos es una matriz cuadrada, con tantas filas y columnas como atributos tengamos. El elemento (i, j) de la matriz indica la correlación que existe entre el atributo i-ésimo y j-ésimo de los contemplados, y los colores significan el grado de correlación que existe entre las variables. Tonos más intensos de azul indican fuerte correlación positiva, mientras que tonos de rojo indican correlación negativa. Es por esto por lo que la matriz es simétrica (la correlación de i con j es igual a la de j con i), y la diagonal tiene un tono azul intenso (todo dato tiene una correlación de 1 consigo mismo). Podemos observar en nuestra tabla que existen fuertes correlaciones entre algunos de nuestros atributos (algunos son el valor absoluto de el otro por lo que la correlación es 1) . El atributo subjetividad sí que tiene cierta correlación con la polaridad de los sentimientos.



Podemos aprovechar la salida de este nodo para filtrar las columnas por correlación lineal. Existe un nodo llamado correlation filter, que tomando como entrada la salida del nodo linear correlation y el conjunto de datos original, da como salida el conjunto de datos quitando las columnas necesarias para evitar la correlación. El valor de correlación a partir del cual se considera que el resultado puede empeorar es configurable en el nodo. En nuestro caso, hemos establecido el valor en 0,5. Los resultados obtenidos tras el preprocesamiento son los siguientes:

Modelo	V positivos	F positivos	V negativos	F negativos	Precisión	Sensibilidad	Especificidad	Valor F
Con preprocesamiento	3201	6981	23737	5725	0.679	0.359	0.773	0.335

Cuadro 15: Resultados con preprocesamiento

Como podemos observar, hemos conseguido mejorar la sensibilidad de forma considerable, aunque a cambio de perder cierta especificidad (esto significa, clasificamos mejor los datos de la clase positiva, a costa de perder capacidad de clasificación respecto de la clase negativa).
Podemos considerar entonces que hemos mejorado los resultados considerablemente ya que a pesar de que el accuracy es algo menor, la capacidad de clasificar correctamente datos de la clase positiva aumenta de forma muy significativa.

Pasamos ahora a comentar el preprocesamiento realizado para kNN.

5.3. Preprocesamiento para el algoritmo kNN

Comentamos el último de los algoritmos que hemos elegido para realizar el preprocesamiento. Intentaremos mejorar los resultados para conseguir un algoritmo competitivo. Lo primero que haremos será cambiar la forma de convertir las variables categóricas.

Veamos los resultados obtenidos al hacer este preprocesamiento:

Modelo	V positivos	F positivos	V negativos	F negativos	Precisión	Sensibilidad	Especificidad	Valor F
Con preprocesamiento	1393	2879	27839	7533	0.737	0.156	0.906	0.211

Cuadro 16: Resultados con preprocesamiento

Podemos observar que el resultado es relativamente mejor, aunque la mejora no es muy significativa. Esto viene explicado por que, aunque es verdad que la conversión de variables nominales a binarias es más lógica que la conversión a variables numéricas, también implica introducir muchas más variables.

Pasamos ahora a interpretar los resultados.

6. Interpretación de resultados

Para interpretar de forma mas sencilla los resultados, presentamos una tabla con los resultados de todos los algoritmos en su versión final y sus respectivas curvas de ROC.

Modelo	V positivos	F positivos	V negativos	F negativos	Accuracy	Sensibilidad	Especificidad	F1-score
Gradient Boosted	558	537	30181	8368	0.775	0.063	0.983	0.111
Naive Bayes	3201	6981	23737	5725	0.679	0.359	0.773	0.335
Árbol de decisión	4935	13829	16889	3991	0.55	0.553	0.55	0.356
Random Forest	229	217	30501	8697	0.775	0.026	0.993	0.049
Red neuronal	830	944	29774	8096	0.772	0.093	0.969	0.155
kNN	1393	2879	27839	7533	0.737	0.156	0.906	0.211

Cuadro 17: Tabla resumen de los resultados obtenidos por los modelos

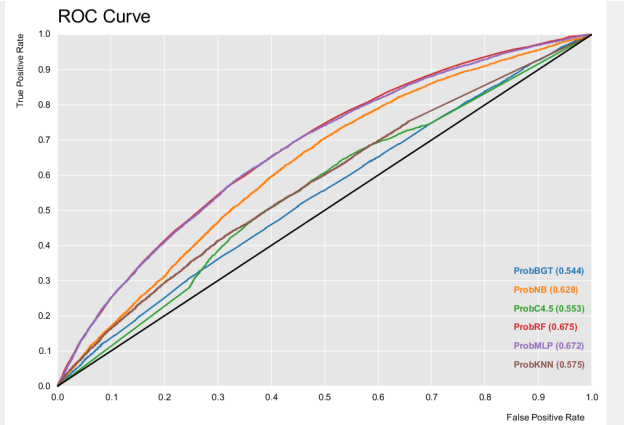


Figura 5: Curva ROC de los Modelos

Hacemos a continuación un análisis general de los resultados obtenidos por nuestros algoritmos para resolver el problema de clasificación al que nos enfrentábamos. Recordamos que nos encontramos ante un conjunto de datos socioculturales, en el que teníamos que clasificar una noticia en internet en función de si será popular o no popular. Se considera como clase positiva aquellas noticias que

son populares.

Uno de los principales problemas con los que teníamos que lidiar era el desbalanceo de las clases, dado que tres cuartas partes de los datos de nuestro conjunto pertenecían a la clase negativa. Esto dificulta, para muchos de los algoritmos contemplados, el aprendizaje para la clasificación de objetos de la clase positiva. Hemos contemplado una metodología para resolver esta problemática, que consiste en hacer un undersampling de la clase mayoritaria. El oversampling es computacionalmente mucho más costoso que el undersampling, por lo que se decide utilizar la segunda metodología con los algoritmos que tienen problemas con la clasificación de clases desbalanceadas.

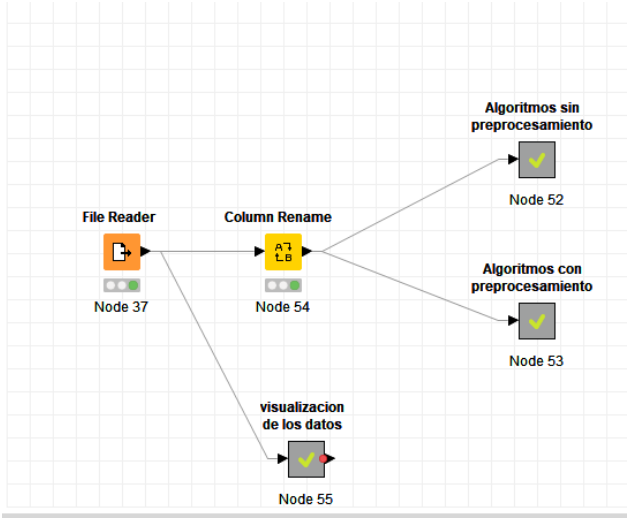
Otro de los problemas afrontados es que existen datos perdidos en el conjunto de datos. Dado que no tenemos información sobre el problema a resolver, se decide resolver esta problemática con el método que a priori introduce menos ruido en el conjunto de datos. Consiste en asignar a los valores numéricos perdidos la media de los valores de dicha característica, y para los valores nominales el valor más repetido.

Una vez hechas estas consideraciones, pasamos a ejecutar los algoritmos con los distintos apartados y a estudiar los resultados obtenidos. Observamos que para Naive Bayes, Random Forest y MLP, el desbalanceo es bastante perjudicial, problema que se arregla al aplicar el preprocesamiento (undersampling). Para los demás algoritmos, el desbalanceo no es tan significativo. En kNN , la conversión de las clases nominales a clases binarias tiene un mejor comportamiento en combinación con la conversión a clases numéricas simples. Esto indica que el preprocesamiento hecho de forma inteligente puede suponer una importante mejoría en los resultados conseguidos después por los algoritmos. Por último, la configuración de los algoritmos nos ha mostrado que, en muchos casos, el aumento de complejidad del algoritmo produce un aumento de tiempo de entrenamiento para el mismo, pero suele venir asociado con una mejora en los resultados. Esto hace que sea interesante la reflexión de qué estrategia utilizamos para resolver nuestro problema, dependiendo de las restricciones que se nos impongan, aunque no podamos hacer esta reflexión en este caso, ya que no nos encontramos resolviendo un problema real. No obstante, en este caso probablemente sería más interesante mejorar los resultados, ya que este conjunto de datos no maneja datos que a priori necesiten restricciones de tiempo críticas.

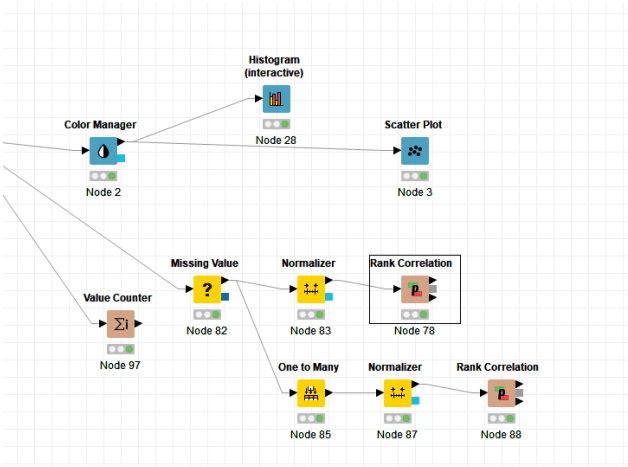
Ahora vamos a intentar, utilizando la información extraída de los modelos, explicar qué variables son realmente más relevantes en la clasificación de individuos. Para ello, exploramos los modelos interpretables que hemos generado. Estos son los modelos basados en árboles (árbol de decisión y random forest) y las tablas que se utilizan para Naive Bayes. Estas últimas son más difíciles de interpretar, pero se puede conseguir información sobre variables de cierta relevancia mirando las frecuencias relativas de cada uno de los valores dependiendo de la clase a la que pertenezca cada objeto.

7. Contenido adicional

En esta sección hemos considerado conveniente mostrar el apartado referente a la visualización de los datos, asi como el flujo de nodos de KNIME.

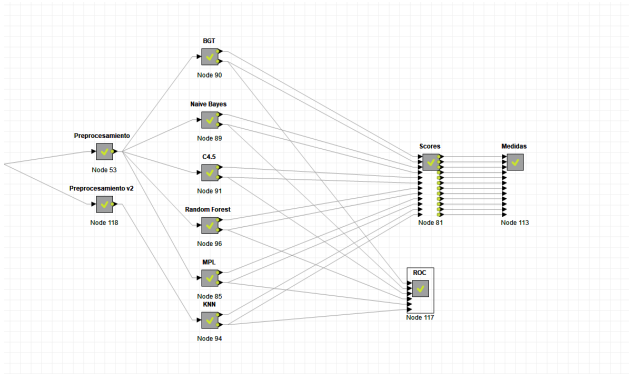


Como podemos observar tenemos 3 metanodos principales. Uno es para los algoritmos sin preprocesamiento, donde solo se encuentran los metanodos de cross validation con sus respectivos scorer, otro para los algoritmos con preprocesamiento y el último para la visualización de los datos.



En el nodo de visualización calculamos un histograma para ver el desbalanceo de clases, así como la correlación entre las caracteristicas

Aqui podemos ver el desbalanceo en cuestión, como hemos comentado a lo largo de la práctica, la cantidad de instancias de la clase negativa es muy superior a la de la positiva



En el metanodo donde incluimos el preprocesamiento, tenemos las distintas versiones de cada algoritmo con su respectivo metanodo. Además, calculamos la tabla y curva ROC de los mejores algoritmos y calculamos las medidas como Gmeasure o Gmean de todos los algoritmos (metanodo medidas). El primer preprocesamiento se diferencia del segundo en que no incluye el nodo one to many.