Universidad de Granada

Inteligencia de Negocio

Práctica 1: Análisis predictivo empresarial mediante clasificación

Miguel Ángel Torres López (Grupo 1)

torresma1996@correo.ugr.es

November 4, 2018

Contents

List of Figures

1 Introducción

1.1 Problema abordado

En esta práctica vamos a estudiar si es posible predecir cuándo una noticia será popular. Para ello vamos a usar un conjunto de datos[1] publicado en UCI, el repositorio de datos para machine learning. Los datos han sido levemente modificados por el profesor de la asignatura, aunque la esencia del problema y los atributos usados son los mismos.

El conjunto de datos esta compuesto por 39644 noticias con 51 atributos cada una. Dentro de los 51 atributos, 50 de ellos son atributos que describen características de la noticia. El atributo restante es de clase binaria y representa si la noticia fue popular o no. Todos los atributos de descripción son numéricos exceptuando uno de ellos, el día de la semana. En ocasiones transformaremos este campo en valores numéricos, para poder aplicar ciertos herramientas de predicción que solo funcionan con valores, como el KNN.

1.2 Datos corruptos

Como puede verse en la figura 1, los datos que se tratan tienen valores perdidos. Esto puede generar algunos errores en ciertos algoritmos que habrá que tener en cuenta.

Row ID	..	S weekday	D LDA_00	D LDA_01	D LDA_02
Row0		wednesday	0.02	0.02	0.02
Row1		?	0.487	0.04	0.04
Row2		saturday	0.02	0.02	0.558
Row3		monday	0.029	0.389	0.349
Row4		wednesday	0.022	0.022	0.546
Row5		friday	0.531	0.035	0.033
Row6		sunday	0.02	0.143	0.02
Row7		saturday	0.509	0.022	0.022
Row8		thursday	0.04	?	0.243
Row9		thursday	0.502	0.029	0.181
Row10		tuesday	0.022	0.022	0.911
Row11		tuesday	0.05	0.306	0.05
Row12		friday	0.239	?	0.04

Figure 1: Valores perdidos en los datos.

Para poder ver mejor la distribución de los valores perdidos y tener en cuenta cuales son los campos afectados por los valores perdidos he usado en KNIME un nodo de filtrado de valores perdidos. He ejecutado el nodo con dos configuraciones distintas, la primera de ellas puede verse en la figura 2.

Cuando filtramos los campos que tienen un 0% de valores perdidos no se devuelve ninguna columna. Esto quiere decir que todas las columnas tienen al menos un valor perdido. En segundo lugar, limito las columnas válidas

a aquellas que tengan como máximo un 1% de valores perdidos. En este caso todas las columnas pasan el filtro. Por tanto aseguramos que todas las columnas tienen entre un 0% y un 1% de valores perdidos.

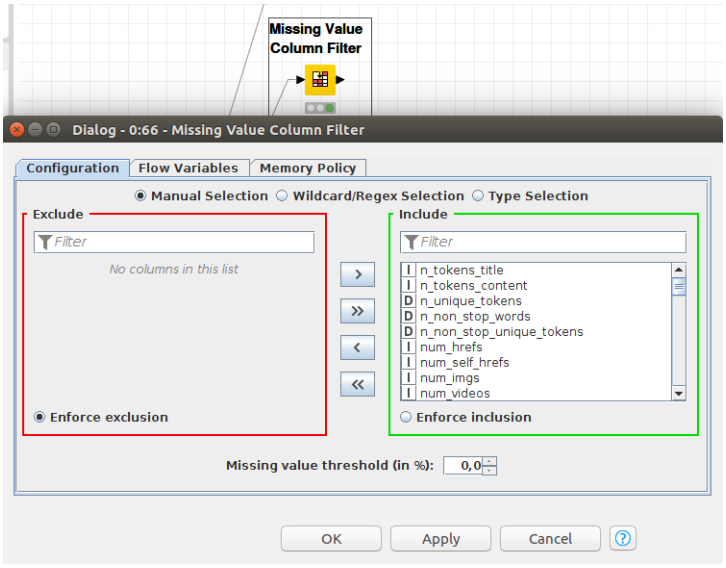

Figure 2: Valores perdidos en los datos.

Otro tipo de valores que podemos tratar como corruptos son los valores extremos. En el conjunto hay valores para algunos atributos que se salen de los valores normales del resto de noticias. Por ilustrar un ejemplo, se puede visualizar la gráfica de la figura 3. En eje horizontal se encuentra la popularidad de la noticia y en el eje vertical el número de tokens únicos de la noticia. Se observa como hay un dato extraviado en la clase de las noticias populares, en concreto, la noticia 28849.

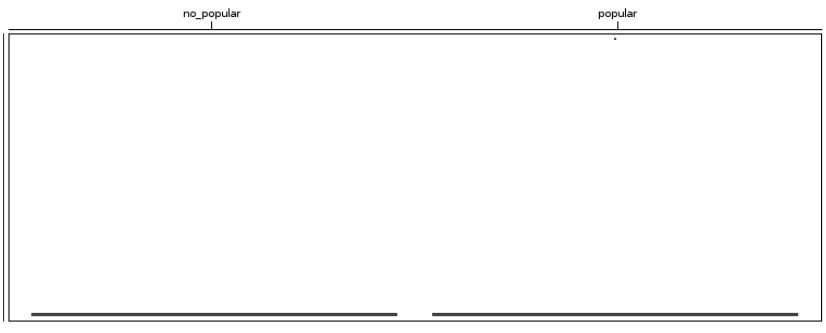

Figure 3: Valores perdidos en los datos.

Claramente, este dato no representa al conjunto de noticias populares, pero su clara identificación podría añadir un peso falso en ciertos procedimientos. Por ejemplo, de cara a realizar un árbol de decisión, se podría crear una rama específica para clasificar bien este caso. Pero esta casuística tan singular no debería ser clasificada, es por tanto necesario prestar atención para disipar la influencia de este tipo de datos.

1.3 Estratificación

El conjunto de las noticias que tratamos esta significativamente estratificado, esto es, existen más noticias no populares que populares. Esto supondrá una tendencia de los algoritmos a colocar los datos en la clase de las noticias no populares. Para evitar esto, previo a la clasificación usaremos el sistema *Cross Validation* con estratificación, como se muestra en la figura 4. Esto hace que se separen los datos en training de forma que los dos valores de popularidad sean igual de frecuentes en el conjunto.

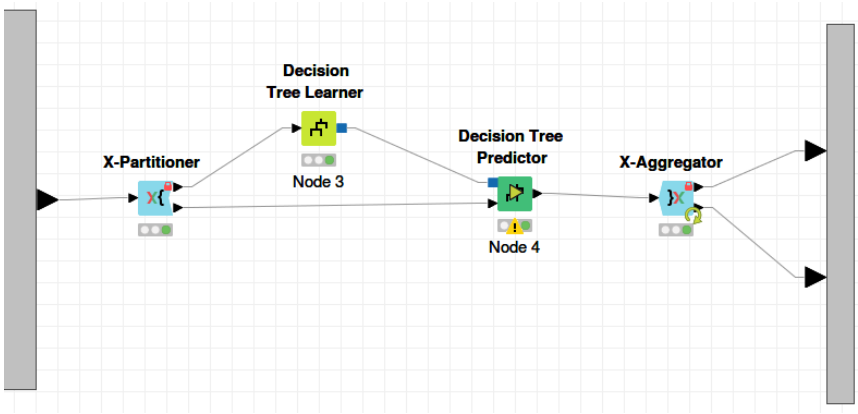

Figure 4: Interior de la estructura *Cross Validation*.

Si no hiciésemos esto, podría darse el caso de que el conjunto de entrenamiento tuviese tan pocos datos de la clase popular que una simple predicción fija diese un alto porcentaje de acierto en *training*. Pero al predecir el conjunto de prueba, habría muchos falsos negativos.

2 Resultados obtenidos

En esta sección se muestran los algoritmos aplicados sobre el problema y su resultado con la mejor configuración encontrada. Otras configuraciones de los algoritmos pueden encontrarse en el apartado 4. Se han aplicado 6 algoritmos distintos.

2.1 Árbol de decisión C4.5

El árbol de decisión C4.5 es considerado uno de los algoritmos base de la ciencia de datos. Es de complejidad sencilla, interpretable y suele dar buenos resultados tanto para datos discretos como continuos. Adems tiene tolerancia frente a pequeñas variaciones en el conjunto de datos y es fácil controlar el sobreaprendizaje aumentando el mínimo de elementos por nodo.

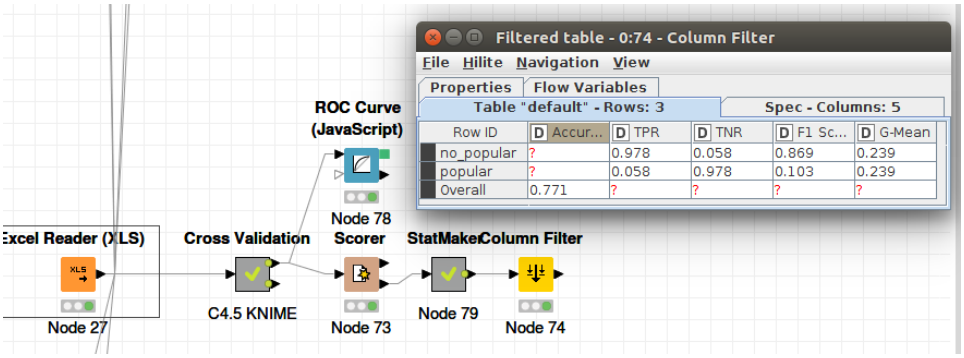

Figure 5: Flujo y resultados del algoritmo C4.5.

En este problema con un mínimo de elementos por nodo de 30 y poda se alcanza un 77.1% de tasa de acierto. Aunque vemos claramente que la mayor parte de los aciertos se producen en la clase de las noticias no populares, mientras que el porcentaje de acierto de las populares es bastante pequeño.

Como puede verse en las estadísticas calculadas, el *True Positive Rate* de la clase no popular está casi a 1, la puntuación máxima, mientras que el de la clase popular es mucho peor. La curva ROC, a pesar de estar lejos de ser buena, es algo mejor que la clasificación aleatoria, aunque, como ya hemos notado, esta bastante influenciado por el buen resultado de las noticias no populares.

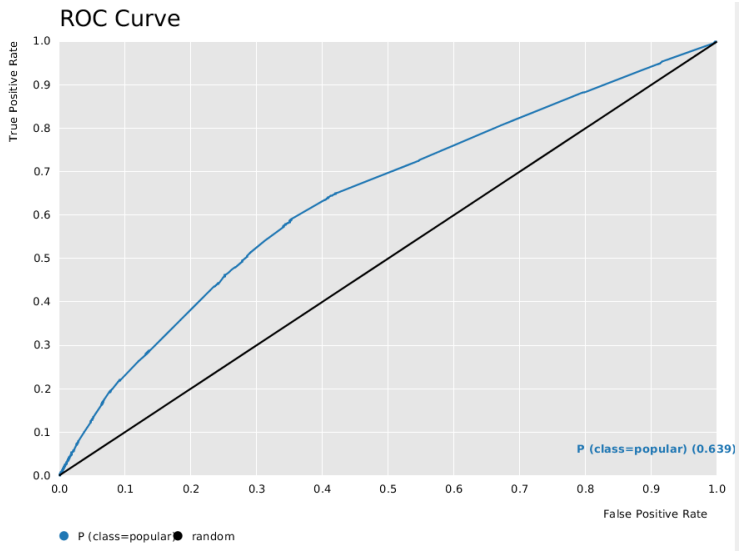

Figure 6: Curva ROC para C4.5.

2.2 KNN

El algoritmo *K-Nearest Neighbour*(KNN) es un método un poco lento pero bastante eficaz cuando la clase a predecir tiene cierto agrupamiento. Aquí hay que notar que tenemos un atributo de cadena de texto que no es la clase. Aunque en teoría se puede calcular la distancia de dos *strings* por semejanza en el teclado o en diccionario, KNIME no tiene soporte para este tipo de cálculos.

En su lugar hay que hacer una transformación del campo a un entero o suprimir dicha columna. He optado por hacer la transformación, a pesar de que esto produce alteraciones en la relevancia del campo. Por ejemplo, la distancia de un lunes(1) a un martes(2) es de uno pero la distancia de un lunes(1) a un domingo(7) es de seis aunque realmente tendría que ser uno.

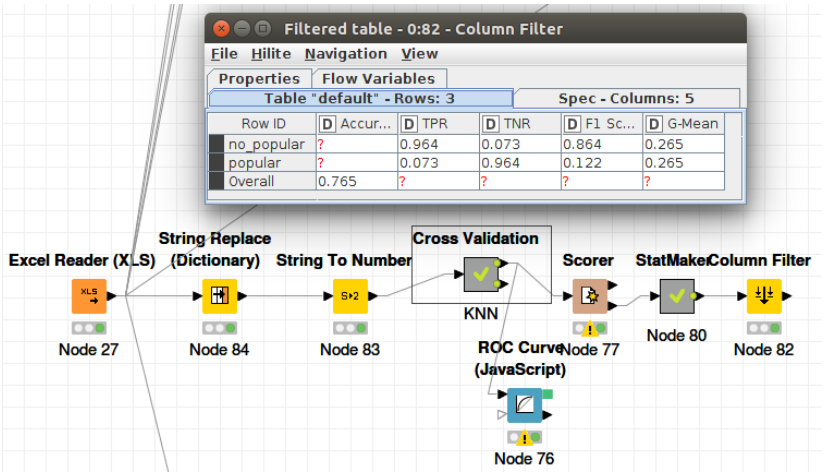

Figure 7: Flujo y resultados del algoritmo KNN.

Los resultados para K=13 con pesos pueden verse en la figura 7. La tasa de aciertos es de 76.5%, pero realmente es bastante mala, pues los aciertos están concentrados prácticamente en su totalidad en la clase de las no populares. En la curva ROC podemos ver que este método, aun siendo bastante pesado, produce un resultado poco mejor que la clasificación aleatoria.

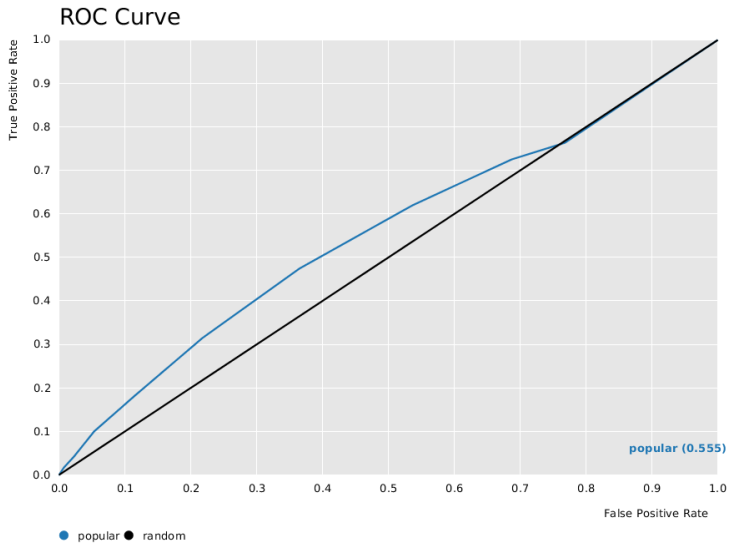

Figure 8: Curva ROC para KNN con K=13.

2.3 Random Forest

Este algoritmo es el método del árbol de decisión C4.5 pero aumentando la complejidad con múltiples árboles construidos con distintos conjuntos de datos. Para predecir la clase final de cada dato, se utiliza una votación por mayoría simple entre todos los árboles resultados del aprendizaje. En cuanto a complejidad temporal, es claramente más pesado que el C4.5, aunque también suele dar mejores resultados que éste. Los resultados del Random Forest con 100 árboles de decisión aparecen en la figura 9.

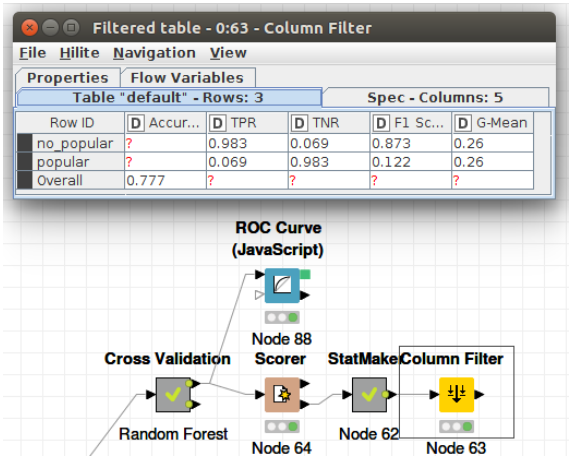

Figure 9: Flujo y resultados del algoritmo RandomForest.

Podemos ver que la tasa de aciertos general esta en 77.7% aunque, como en el resto de casos, los aciertos en el caso de las noticias populares es bastante bajo. No obstante, tanto el *TPR* como el *F1 Score* han mejorado unas décimas, esto es síntoma de que este algoritmo tiene mejores resultados. Como refuerzo de esta idea encontramos la curva ROC, que en esta ocasión llega a tener 0.7 puntos de área.

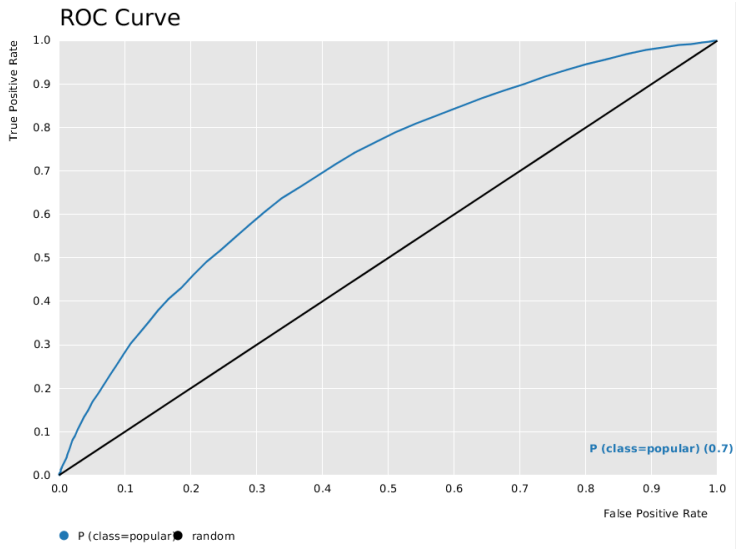

Figure 10: Curva ROC en el RandomForest.

2.4 AdaBoost

Este algoritmo se basa en la utilización de algoritmos auxiliares más sencillos para posteriormente unir sus respectivas soluciones con un sistema iterativo de pesos en el que los datos mal clasificados pesan más que el resto. Es por tanto uno de los algoritmos más rápidos, siempre que se use con algoritmos auxiliares poco complejos en tiempo. En la figura 11 podemos ver el flujo y el resultado para la mejor combinación encontrada (se explica en la sección 4).

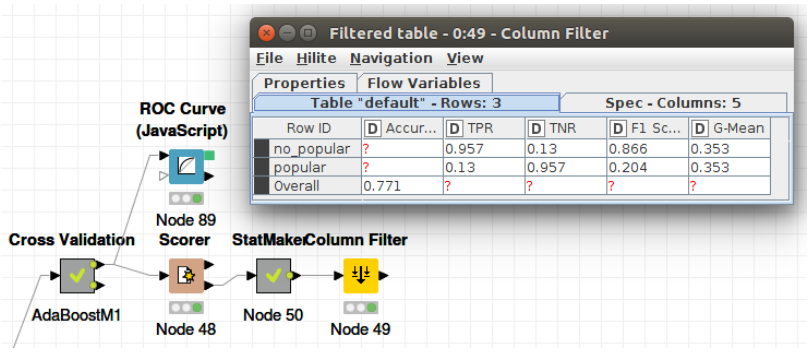

Figure 11: Flujo y resultados del algoritmo AdaBoost.

Con este algoritmo la tasa de acierto es un poco más balanceada, aunque el valor del AUC es un poco inferior que en el caso del Random Forest. Mientras que el Random Forest alcanzaba los 0.7 puntos, ahora nos quedamos en

los 0,692. A priori puede parecer una diferencia insignificante, pero cuando se trata de clasificar miles de casos si que es notable.

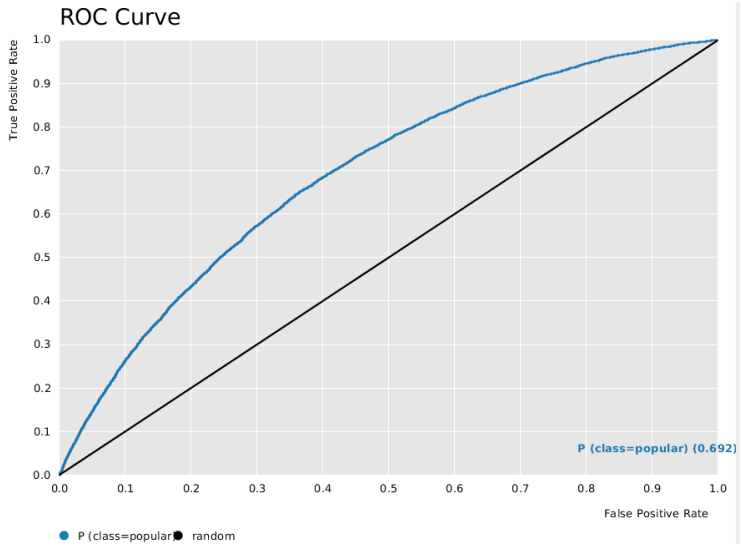

Figure 12: Curva ROC en el AdaBoost.

2.5 Perceptrón Multicapa

Para esta sección se requiere un poco más de preparación de datos. El perceptron solo admite datos cuyos atributos sean doubles normalizados entre 0 y 1. En nuestros datos tenemos Integer, que no son difíciles de pasar a double, y strings, que tenemos que transformar con cuidado.

Tenemos dos campos de tipo string. El primero, la clase a predecir, es bastante sencillo de transformar a double. Basta remplazar *popular* por 1 y *no_popular* por 0, luego con el nodo *String to Number* de KNIME completaremos nuestro objetivo. Para el segundo campo, el día de la semana, tenemos que usar un diccionario, que para cada valor le haga corresponder un número del 1 al 7. Ya en el algoritmo KNN se comenta el problema que esto conlleva, la distancia de lunes a martes debería ser la misma que de domingo a lunes, pero no es así. El flujo puede verse en la figura 13

Otra opción para transformar el día de la semana es usar el nodo *One to Many*, que nos divide el campo en 7 campos binarios. Se verán sus resultados en el apartado de procesado de datos.

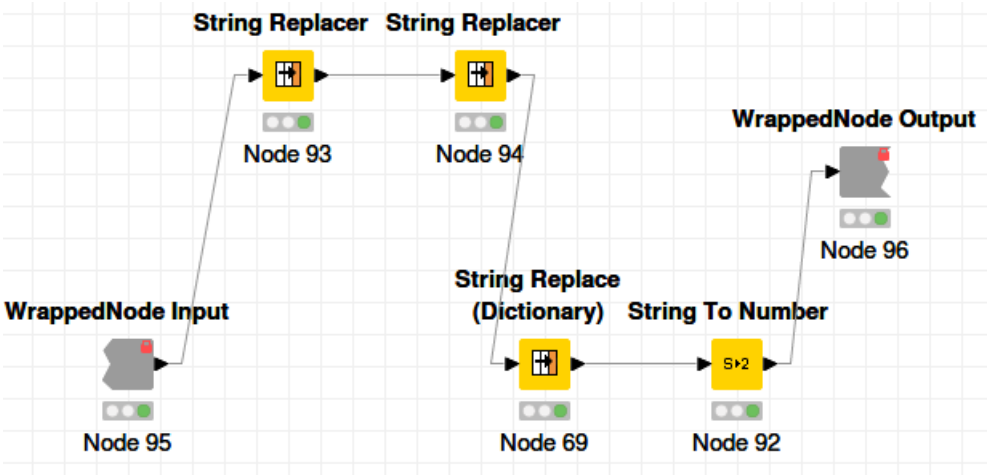

Figure 13: Flujo para cambiar strings a doubles.

El flujo para hacer un *Cross Validation* es similar a los anteriores, puede verse en la figura 14. Además, debido a que la red neuronal necesita multiplicar y sumar los valores de los campos, estos no pueden ser nulos. Para sortear este obstáculo, los datos de entrenamiento los filtro para que solo entren datos sin valores perdidos. En el conjunto de test no puedo proceder igual, pues estaríamos quitando datos, difíciles de clasificar, y las comparaciones con el resto de algoritmos no serían justas. En lugar de esto, añado un nodo *Missing Value* para imponer valores.

Es aconsejable normalizar los datos entre 0 y 1 para que los campos tengan la misma importancia de partida dentro de la red neuronal.

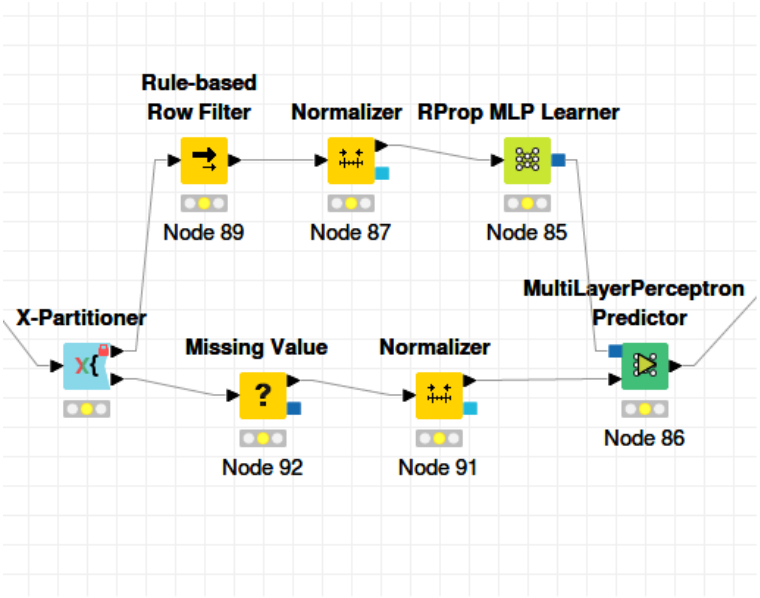

Figure 14: Flujo para cambiar strings a doubles.

La salida de una red neuronal es un poco más incómoda que la del resto de algoritmos. Para cada dato, se devuelve una valor de 0 a 1 que, en nuestro caso, significa la probabilidad de ser de la clase *no_popular*(0) o *popular*(1). El problema está en que en algunas instalaciones de KNIME no se pone de forma explicita si un dato se clasifica de una clase o de otra. Tras observar la salida e ir probando distintos valores, voy a considerar que por encima del valor 0.45 es una noticia popular y por debajo, no popular. Aunque he podido comprobar que en otras versiones, por defecto si se especifica la clasificación y su valor es 0.5.

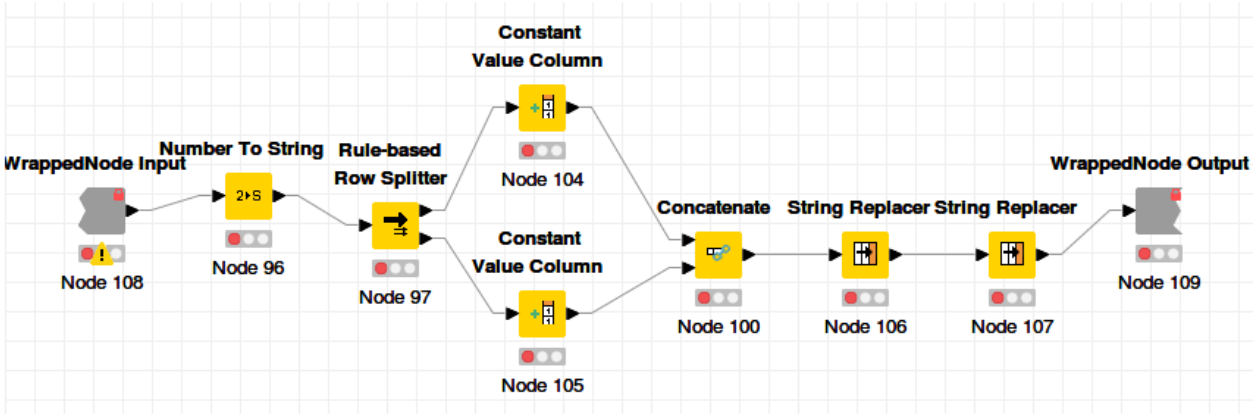

Figure 15: Flujo para cambiar la salida.

Para poder ver las estadísticas de igual forma que en el resto de algoritmos, es necesario transformar manualmente la predicción con algunos nodos. En la figura 15 puede verse el flujo de KNIME para transformar la salida. En resumen lo que se hace es escribir la clase predicha y volver a pasar la clase verdadera de double a string.

3 Análisis de resultados

En la figura 16 se comparan las mejores configuraciones para cada algoritmo. Debemos aclarar que cuando hablamos de mejor configuración no nos estamos refiriendo a la que más aciertos globales tiene, si no a la que mejor clasifica en proporción a las dos clases. Los datos están claramente desvalanceados y los algoritmos tienden a clasificar todas las noticias como no populares, que es la clase mayoritaria. Es por esto que vamos a dar más relevancia a las configuraciones que mejor balance de aciertos tenga entre sendas clases.

Otras configuraciones pueden verse en la sección 4.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
Decision Tree Sin poda, mínimo 20 elementos por nodo	75,70%	0,129	0,939	0,193	0,348	0,64
KNN K=5, con pesos	74,20%	0,154	0,912	0,211	0,375	0,546
Random Forest 50 árboles	77,70%	0,051	0,989	0,094	0,224	0,696
AdaBoost 60 iteraciones, con resampling, Random Tree	77,10%	0.13	0,957	0,204	0,353	0,692
Perceptrón Multicapa 200 iteraciones, 1 capa, 51 neuronas por capa	75,00%	0,293	0,882	0,345	0,509	0,682

Figure 16: Mejores resultados para cada algoritmo.

3.1 Predicciones desbalanceadas y tasa de acierto global

Como ya hemos notado, las predicciones de todos los métodos tienden a clasificar las noticias como no populares. En la sección 4 se ve cómo cuando se provoca sobreaprendizaje sobre el conjunto de entrenamiento los algoritmos marcan las noticias en su mayorá como no populares. Para ver este hecho basta fijarse en el *True Positive Rate (TPR)*, nunca es superior a 0.3. De hecho, solo hay un único método que casi acierta el 30% de las noticias populares, el perceptrón multicapa. El resto están bastante lejos de este resultado.

El motivo de este "éxito" podría radicar en la ventaja de las redes neuronales de excluir ciertos parámetros de entre los dados. Es fácil que una neurona acabe tomando como peso de una de sus entradas un 0, provocando la anulación de este resultado. Por el contrario, el algoritmo KNN est'a obligado a tener en cuenta todos los valores y el resto de algoritmos(basados en rboles) intentan coger el máximo número de atributos.

De forma casi opuesta al balanceo de acierto tenemos la tasa de acierto global. Esta tasa mejora cuando el *F1 Score* baja. Realmente lo que está ocurriendo es que las noticias estan siendo clasificadas como no populares, que son las más abundantes en nuestro conjunto de datos (aproximadamente un 75%). Esto sumado a algunos aciertos en la clase popular hacen que se obtenga hasta un 77.7% de aciertos globales. Entonces el problema, además de que es una tasa bastante baja, es la procedencia de estos aciertos.

3.2 Curvas ROC

Para ver que algoritmo tiene más acierto según la probabilidad a partir de la cual clasifiquemos como clase no popular se usa la curva ROC. En el apartado anterior hemos visto la curva de cada algoritmo, pero para una mejor comparación podemos unir todas las curvas en un solo gráfico dentro de KNIME. En la figura 17 se aprecia que algoritmo obtiene mejor resultado.

Por un lado, el KNN obtiene los peores resultados, llegando ha estar en probabilidades altas por debajo de la clasificatoria aleatoria. Es el claro perdedor de momento. Y digo de momento porque en la sección 5 hablaremos de como se comportan los algoritmos tras el procesamiento de los datos con breves transformaciones.

El siguiente algoritmo es el árbol de decisión C4.5, similar a los tres siguientes pero decae en probabilidades altas. Esto es un indicador de que en noticias que clasifica con bastante probabilidad como no populares son realmente populares. Es decir, comete fallos graves.

Por último, y bastante parejos, estn los otros tres. El perceptrón es el peor de estos tres, random forest comienza con algo más de acierto pero en pero en probabilidades altas el mejor es el algoritmo Adaboost.

Dar una respuesta a por qué los algoritmos se ordenan así es bastante complejo. Además, tampoco hay diferencias demasiado grandes, excepto en el KNN y en los algoritmos basados en árboles, que nos hagan pensar que hay alguna razón distinta al propio azar.

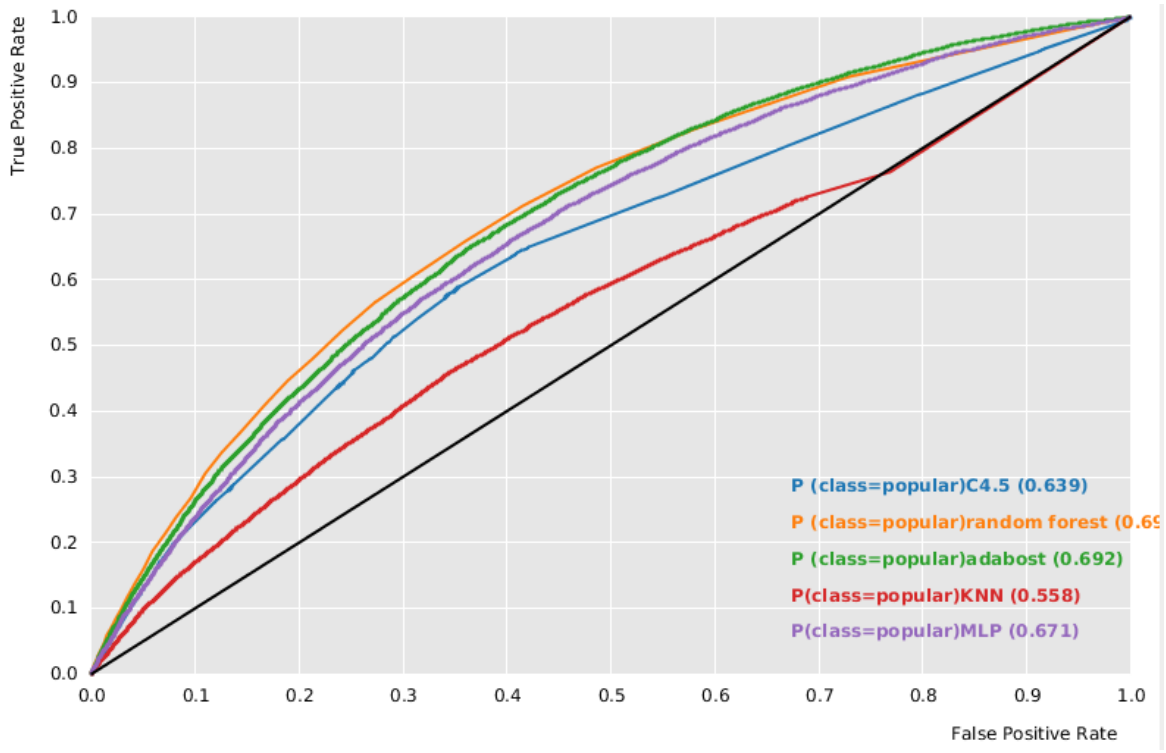

Figure 17: Comparación de las curvas ROC.

El caso del KNN si es bastante claro. Este algoritmo es muy sensible a los rangos de valores de los parámetros. Para paliar este hecho, se podría optar por normalizar todos los parámetros para que las distancias generadas en el algoritmo sean más representativas. Esto, junto con la eliminación de algunos atributos un poco difíciles de normalizar, como el día de la semana, podría mejorar los resultados. Esto lo veremos en la sección 5 de procesamiento.

La relación entre el C4.5 y el random forest también se puede presumir. Ya que el primero es un árbol y el segundo es una votación ponderada de muchos árboles. Por tanto podemos asumir que el algoritmo random forest tendrá una mayor tasa de acierto.

4 Configuración de algoritmos

4.1 Configuración del árbol de decisión C4.5

Para configurar el árbol de decisión C4.5 consideramos dos parámetros principales, la poda y los elementos mínimos por nodo. Para entender bien los resultados que se ilustran en la figura 18 primero debemos entender bien lo que hacen sendos parámetros.

- **Poda**. La poda activada produce que una vez terminado el algoritmo, se contraigan los nodos del árbol si la cantidad de elementos en el nodo es baja. Esto produce que algunos casos especiales en los datos no se tengan en cuenta a la hora de clasificar. Se sintetiza el árbol asumiendo una pequeña tasa de error en los datos de entrenamiento en pos de una mejor generalización para los datos de test.

  En este caso, las noticias populares deben ser bastante difíciles de separar de las no populares, es decir, las noticias populares no tienen un atributo que las diferencie especialmente. Esto produce que al efectuar la poda se asuman demasiadas generalizaciones de hojas poco frecuentes asociadas a noticias populares. Podemos ver que el *F1 Score* es algo más alto en las configuraciones sin poda, aunque los aciertos globales bajan al permitir el sobreaprendizaje.

  El hecho de que los nodos hoja de los árboles asociados a noticias populares tengan tan pocos elementos podría ser a causa de la poca diferencia entre los valores de una clase y la otra para cada atributo. De hecho, es posible que valores similares de atributos produzcan noticas de sendas clases. Veremos de nuevo esta patología de los datos en la sección 6, donde discutiremos de forma más profunda la separación de los datos.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
Poda MDL, min 5 elementos por nodo	77,40%	0,013	0,995	0,024	0,112	0,627
Poda MDL, min 30 elementos por nodo	77,10%	0,058	0,978	0,103	0,239	0,639
Poda MDL, min 50 elementos por nodo	77,20%	0,055	0,98	0,097	0,232	0,638
Sin poda, min 40 elementos por nodo	76,00%	0,112	0,94	0,174	0,326	0,651
Sin poda, min 20 elementos por nodo	75,70%	0,129	0,939	0,193	0,348	0,64

Figure 18: Resultados para distintas configuraciones del algoritmo C4.5.

- **Elementos mínimos por nodo**. En este parámetro podríamos decir que hay un número óptimo de elementos que nos da el mejor resultado.

  Por un lado, subir esta cantidad provocaría una generalización demasiado grande, aunque en este caso con las noticias no populares más abundantes provoca que el porcentaje de aciertos globales suba. No obstante, podemos ver como las métricas orientadas a medir el porcentaje de acierto en sendas clases, la clase de las populares pierde precisión.

  Por otro lado, si bajamos esta cantidad, estaríamos facilitando la aparición de hojas muy especializadas que son incapaces de predecir la clase para valores no aprendidos. Bajan el *F1 Score* y el área bajo la curva ROC.

4.2 Configuración del KNN

Este algoritmo no tiene muchos parámetros que ajustar. De hecho, solo hay dos cambios posibles, modificar el valor de K, o número de vecinos a considerar, y activar o desactivar la ponderación de los vecinos por su distancia.

- **Número de vecinos K**. Este valor modifica la cantidad de vecinos que se buscan cuando se quiere clasificar un dato. A mayor número de vecinos, más pesado será el cálculo pero la clasificiación se hace más segura de existir entornos de datos separables.

  Con este conjunto de datos, se observa en la figura 19 que la tasa de aciertos global sube, pero se debe a que se clasifican la mayor parte de los datos como no populares. Esto puede deberse a que las noticias

populares no son fáciles de separar en clusters multitudinarios, si no que están en dispersos o, en el mejor de los casos, en agrupaciones muy pequeñas. Aparecerá más adelante esta patología de los datos en la sección 6, donde discutiremos de forma más profunda la separación de los datos.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
K=5, sin pesos	74,50%	0,141	0,919	0,198	0,36	0,544
K=13, sin pesos	76,70%	0,057	0,972	0,098	0,234	0,555
K=25, sin pesos	77,30%	0,027	0,988	0,051	0,163	0,562
K=5, con pesos	74,20%	0,154	0,912	0,211	0,375	0,546
K=13, con pesos	76,50%	0,073	0,964	0,122	0,265	0,558

Figure 19: Resultados para distintas configuraciones del algoritmo KNN.

- **Ponderación por distancia**. Cuando la ponderación está desactivada, la elección de la clase de un dato se hace por votación con mayoría simple. Activando este parámetro los votos se ponderan en función de la distancia. Por tanto, activado aumenta ligeramente el tiempo de ejecución, pero mejor en cierto sentido la predicción. Si por ejemplo, tenemos dos noticias populares con valores muy cercanos, el algoritmo clasificará correctamente. Si por el contrario, los datos de sendas clases están muy cerca unos de otros, podría ocurrir que estuviesemos dando un peso a la clase contraría.

  Una vez más, dependemos de que los datos de sendas clases no tengan inconsistencias. De lo contrario este algoritmo hará muchas predicciones de forma incorrecta, y así ocurre.

4.3 Configuración del Random Forest

El algoritmo Random Forest no es demasiado manipulable. Podemos escoger el número de árboles con el que se ejecutará y configurar algunos parámetros de estos. Para el número de árboles se presenta una sucesión creciente del valor, para la configuración de los árboles de decisión se escoge la configuración que mejor resultado dio en el algoritmo de árbol de decisión C4.5.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
50 Árboles	77,70%	0,051	0,989	0,094	0,224	0,696
100 Árboles	77,70%	0,047	0,99	0,086	0,215	0,704
200 Árboles	77,80%	0,046	0,99	0,085	0,213	0,709
300 Árboles	77,80%	0,043	0,991	0,081	0,208	0,71

Figure 20: Configuraciones del algoritmo Random Forest.

A priori parece que a medida que usamos más árboles el resultado del algoritmo debería mejorar. Aunque en este conjunto de datos no es así. La tasa de aciertos globales disminuye cuando disminuimos los árboles, pero conseguimos que los aciertos estén un poco más repartidos entre las dos clases.

El motivo de este hecho es simple, cada dato que pertenece a una clase tiene valores parecidos o cercanos a varios datos de la otra clase. Por lo tanto, en la mayoría de los árboles estos datos, aun siendo de clases contrarias, caerán en las mismas hojas. Sin embargo, a la vista del TPR y TNR vemos que la clase no popular si se clasifica bien, es decir, estas hojas se toman como no populares. Entonces, podríamos preguntarnos por qué los nodos se etiquetan en la mayor parte de ocasiones como la clase no popular.

4.4 Configuración del AdaBoost

A pesar de que hay más parámetros manipulables, aquí consideramos tres de ellos, las iteraciones, el remplazo y el algoritmo auxiliar.

- **Iteraciones**. En AdaBoost las iteraciones marcan las veces que el algoritmo auxiliar tiene que ejecutarse con los datos ponderados de la iteración anterior. Se puede ver en la figura 21 que en todos los casos el aumento de iteraciones sube el AUC y el *F1 Score*. Esto es bueno, pues nuestro problema principal es clasificar las populares como populares.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
10 iteraciones, sin resampling, DecisionStump	77,40%	0,018	0,993	0,034	0,133	0,674
40 iteraciones, sin resampling, DecisionStump	77,40%	0,04	0,997	0,074	0,199	0,694
10 iteraciones, sin resampling, RandomTree	77,10%	0,054	0,979	0,097	0,213	0,676
60 iteraciones, sin resampling, RandomTree	77,10%	0,094	0,967	0,156	0,302	0,687
60 iteraciones, con resampling, RandomTree	77,10%	0,13	0,957	0,204	0,353	0,692

Figure 21: Resultados para distintas configuraciones del algoritmo AdaBoost.

- **Algoritmo auxiliar**. Supone la pieza central del algoritmo, pues es en última instancia quien decide como se clasifician los datos. Para ilustrar los cambios que produce, he realizado pruebas con dos algoritmos auxiliares. El primero, DecisionStump, es un árbol de decisión binario con un solo nodo, es decir, divide los datos con una línea en dos secciones. El segundo es un Random Tree con profundidad máxima 3, sin poda y con 40 elementos mínimos en cada nodo.

  El algoritmo AdaBoost da mejores resultados cuando su algoritmo auxiliar es más complejo, eso si, también aumenta en complejidad de tiempo.

- **Remplazo**. Si el remplazo esta activado, AdaBoost escogerá los datos para realizar la ejecución con remplazo. Activarlo mejora tímidamente la curva ROC y los aciertos para la clase popular, aunque empeora la otra.

4.5 Configuración del Perceptrón Multicapa

Uno de los parámetros que podríamos ajustar es el valor a partir del cual se clasifica como clase popular. Pero, como ya se mencionó, tras probar varios valores se observa que 0,45 es el óptimo.

Suprimiendo este valor, nos quedan otros 3 valores que podemos manipular: número de iteraciones, número de capas y cantidad de neuronas por capa.

- **Iteraciones**. Corresponde con la cantidad de veces que se ejecutan los datos sobre la red neuronal con su correspondiente ajuste de pesos en las neuronas. Al aumentar el número de iteraciones podemos ver como

se ajustan mejor los datos de la clase no popular mientras disminuyen los de la popular. Por eso, optaremos por un valor bajo.

- **Capas**. Con este valor modificamos el número de capas de neuronas que componen la red neuronal. Un valor alto provocará que la complejidad del algoritmo aumente demasiado y el tiempo de ejecución suba.

- **Neuronas**. Modifica la cantidad de nodos o neuronas que hay por capa. Aumentar este número, aumenta la complejidad considerablemente, aunque permite que el modelo haga más combinaciones en la red neuronal.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
200 iteraciones, 1 capa, 16 neuronas por capa	73.4%	0,293	0,862	0,331	0,502	0,662
100 iteraciones, 1 capa 16 neuronas por capa	76,00%	0,211	0,919	0,284	0,441	0,681
100 iteraciones, 2 capas 16 neuronas por capa	74,30%	0,275	0,88	0,326	0,492	0,663
200 iteraciones, 2 capa, 16 neuronas por capa	73,80%	0,265	0,876	0,313	0,482	0,661
100 iteraciones, 1 capa, 32 neuronas por capa	75,50%	0,254	0,901	0,318	0,478	0,667
100 iteraciones, 1 capa, 51 neuronas por capa	73.4%	0,31	0,852	0,341	0,514	0,665
200 iteraciones, 1 capa, 51 nueronas por capa	75,00%	0,293	0,882	0,345	0,509	0,682

Figure 22: Resultados para distintas configuraciones del algoritmo Perceptrón Multicapa.

Extraemos de los resultados que aumentar el número de capas empeora el modelo y aumentar el número de iteraciones hace que se le de un peso favorable a las noticias no populares. Por intuición, he hecho un test en el que el número de neuronas por capas es igual al número de atributos, consiguiendo las mejores estadísticas del algoritmo.

5 Procesado de datos

5.1 Tratamiento de los valores perdidos

A la hora de tratar los valores desconocidos tenemos generalmente tres métodos.

El primero y más sencillo consiste en eliminar las noticias que contengan valores desconocidos para entrenar los algoritmos. En la misma línea podemos intenar ejecutar los algoritmos eliminando los atributos que contengan un mayor número de valores perdidos. El último, en lugar de eliminar, intenta inferir un valor artificial para ese campo perdido. Hay múltiples formas de elegir el nuevo valor, a continuación se proponen dos.

5.1.1 Eliminación de las filas incompletas

Cuando hablamos de eliminación de filas, debemos tener en cuenta que, en este punto del análisis, no podemos tomar el conjunto de datos y eliminar las noticias que contengan valores perdidos. Ya que esto supondría eliminar datos para el conjunto de test y las comparaciones con el resto de ensayos no serían justas.

En lugar de esto, se deben eliminar las filas con valores perdidos del conjunto de entrenamiento. De esta forma, el modelo se aprenderá solo de datos reales y completos. Aunque luego se intente predecir la clase de datos con valores perdidos.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
Decision Tree Sin poda, mínimo 20 elementos por nodo	75,70%	0,129	0,939	0,193	0,348	0,64
Decision Tree Sin poda, mínimo 20 elementos por nodo Datos tratados	74,30%	0,192	0,903	0,251	0,416	0,615

Figure 23: Resultado del C4.5 con eliminación de filas.

En la figura 23 podemos ver los resultados del C4.5 con la eliminación de filas. Podemos ver que el *TPR* aumenta ligeramente, síntoma de que la clase popular se predice un poco mejor. Sin embargo, baja la tasa global de aciertos y empeora la curva ROC. Los resultados no son mucho mejores que en el caso sin tratamiento de datos.

Un posible motivo de esto, es que los valores quitados en el aprendizaje serían mayoritariamente de la clase no popular. Por tanto, la clase popular

adquiere un poco más de relevancia. No obstante, al disminuir la cantidad de datos del entrenamiento, también disminuimos la calidad del modelo.

5.1.2 Eliminación de las columnas

Siguiendo la idea del punto anterior, podemos intentar esta vez quitar las columnas con datos perdidos. Como hemos visto en la sección introductoria de la práctica, todas las columnas tienen más de un 0% de valores perdidos pero menos de un 1%.

Por tanto, vamos a tener que quitar solo las columnas con mayor tasa de valores perdidos, ya que no podemos quitarlas todas.

En KNIME podemos usar el nodo *Missing Value Column Filter* para filtrar las columnas con menos valores perdidos. Situando el corte en un 0.5%, nos quedamos con 24 columnas de las originales 51. En la figura 24 podemos ver el resultado para dos algoritmos.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
Decision Tree Sin poda, mínimo 20 elementos por nodo	75,70%	0,129	0,939	0,193	0,348	0,64
Decision Tree Sin poda, mínimo 20 elementos por nodo Datos tratados	76,00%	0,082	0,957	0,134	0,281	0,638
KNN K=5, con pesos	74,20%	0,154	0,988	0,221	0,375	0,546
KNN K=5, con pesos Datos tratados	73,10%	0,17	0,894	0,222	0,39	0,566

Figure 24: Resultados quitando columnas para C4.5 y KNN.

En las estadísticas se ve reflejado como el árbol de decisión C4.5 tiene menos condiciones para imponer en los nodos y, en consecuencia, hace peores particiones de los datos en cada rama. El porcentaje de aciertos globales aumenta potenciado por la clasificación mayoritaria de no populares, que suponene mayor porcentaje.

En contraposición, el algoritmo KNN mejora tímidamente. Aumenta el balanceo de aciertos entre las dos clases, pero disminuye el total de aciertos. Esto puede explicarse por el funcionamiento interno del KNN, la distancia. Al quitar los campos con más valores perdidos, hemos mejorado el significado de la distancia entre las noticas.

5.1.3 Introducción de valores frecuentes

Una de las formas de intentar mejorar el rendimiento de los algoritmos es rellenar los valores vacíos con el valor más frecuente, en caso de cadenas de texto y enteros, o la media, en caso de doubles. En la figura 25 podemos ver el sencillo flujo de KNIME para ello.

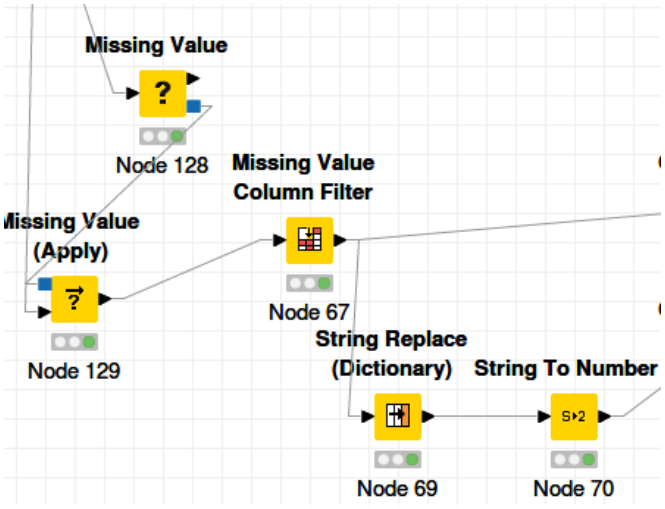

Figure 25: Salida superior para C4.5, inferior para KNN.

Aunque a priori este tratamiento puede parecer buena idea, en la práctica no suele funcionar bien. Si lo pensamos, sustituir los valores vacíos por el valor más frecuente o por la media, implica trasladar los datos alrededor del valor más cénctrico. Esto provocará que los datos se asemejen más y las distancias entre ellos sean menores, dificultando así la caracterización de las clases.

Para ilustrar el empobrecimiento de los datos con este tratamiento, en la figura 26 se muestran los resultados de la mejor configuración del C4.5 y del KNN con este tratamiento.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
Decision Tree Sin poda, mínimo 20 elementos por nodo	75,70%	0,129	0,939	0,193	0,348	0,64
Decision Tree Sin poda, mínimo 20 elementos por nodo Datos tratados	73,40%	0,218	0,884	0,27	0,439	0,613
KNN K=5, con pesos	74,20%	0,154	0,988	0,221	0,375	0,546
KNN K=5, con pesos Datos tratados	74,10%	0,149	0,913	0,205	0,368	0,582

Figure 26: Resultados sin y con introducción de valores frecuentes.

Tanto el C4.5 como el KNN empeoran resultados. En el primero, parece que el balanceo de acierto en clases mejora, aunque la tasa global de aciertos y el área bajo la curva ROC bajan. Si solo bajase la tasa global sería un resultado a considerar pero, al bajar el AUC, podemos ver que el resultado es más pobre.

5.1.4 Introducción de valores aleatorios

En un intento de mejorar el fallo del punto anterior, vamos a introducir, con el mismo flujo de KNIME, nuevos valores de forma aleatoria en los valores desconocidos. Esto lo podemos conseguir rellenando cada campo con el valor de la fila anterior.

Podríamos pensar que estamos introduciendo valores falsos, y así es, que van a eliminar la relación implicita que existe en los datos. No obstante, también es una forma de disuadir a los algoritmos para que no se centren en categorizar los datos por este campo.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
Decision Tree Sin poda, mínimo 20 elementos por nodo	75,70%	0,129	0,939	0,193	0,348	0,64
Decision Tree Sin poda, mínimo 20 elementos por nodo Datos tratados	73,60%	0,217	0,886	0,27	0,439	0,613
KNN K=5, con pesos	74,20%	0,154	0,988	0,221	0,375	0,546
KNN K=5, con pesos Datos tratados	74,10%	0,149	0,913	0,206	0,369	0,584

Figure 27: Resultados sin y con introducción de valores aleatorios.

La diferencia con el apartado anterior es casi imperceptible. El árbol de decisión mejora unas centésimas en algunas estadísticas y, por su parte, el KNN empeora en la misma proporción. En este caso, introducir los valores de forma aleatoria en lugar de introducir el más frecuente no crea una gran diferencia.

Probablemente, este síntoma se debe a la baja proporción de valores perdidos que hay en el conjunto en relación con la cantidad de atributos de cada noticia (Ver aparatado 1.2).

5.2 Desdoblamiento de columnas

En el caso de atributos discretos que tengan un bajo número de posibilidades, en nuestro caso el día de la semana, se puede desdoblar la columna en tantas como valores posibles haya. Esto ayuda a eliminar la dependencia creada en las columnas que se traducen de string a double para ser utilizadas en algoritmos que lo requieren, en nuestro caso el Perceptrón Multicapa. Para realizar esta tarea se usa el nodo *One To Many*.

Como ventaja adicional tenemos que los valores perdidos del día de la semana ya no son un problema, ya que se transforma en ceros en todas las columnas.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
Perceptrón Multicapa 200 iteraciones, 1 capa, 51 neuronas por capa	75,00%	0,293	0,882	0,345	0,509	0,682
Perceptrón Multicapa 200 iteraciones, 1 capa, 51 neuronas por capa Datos tratados	73,90%	0,31	0,864	0,348	0,517	0,672

Figure 28: Resultados sin y con desdoblamiento del día de la semana.

El desdoblamiento de columnas en la red neuronal funciona relativamente bien. Aumenta la tasa de aciertos en la clase popular pero disminuye bastante los aciertos en la otra clase. La curva ROC empeora unas décimas aunque los aciertos estan un poco más balanceados

5.3 Normalización de los datos

Es habitual que los algoritmos estén algo condicionados por los diferentes dimensiones de los valores que puede tomar cada atributo. Por este motivo puede ser conveniente normalizar los datos para que todos los atributos sean igual de importantes.

En KNIME puede hacerse con el nodo *Normalizer*, no se incluye el flujo por su sencillez.

	Aciertos Globales	TPR	TNR	F1 Score	G-Mean	AUC
AdaBoost 60 iteraciones, con resampling, RandomTree	77,10%	0,13	0,957	0,204	0,353	0,692
AdaBoost 60 iteraciones, con resampling, RandomTree Datos tratados	77,10%	0,123	0,959	0,194	0,343	0,692
KNN K=5, con pesos	74,20%	0,154	0,988	0,221	0,375	0,546
KNN K=5, con pesos Datos tratados	73,90%	0,142	0,913	0,198	0,361	0,566

Figure 29: Resultados sin y con normalización de datos.

En la figura 29 podemos ver los resultados después de normalizar los datos. En la curva ROC podemos ver cierta mejoría en sendos casos. El problema está en que esta se ha producido por el acierto de más noticias no populares, justo lo contrario de como nos gustaría.

6 Interpretación de resultados

6.1 Inconsistencia

Como ya hemos visto anteriormente, los datos parecen tener cierta inconsistencia o, al menos, no son distinguibles en clases. Para ilustrar esta idea, he realizado varias gráficas.

La primera de ellas, muestra las matrices de valores que enfrentan a cada LDA con otro. En color rojo se marcan las noticias no populares y en azul las populares. Notamos que no parece haber ninguna sección evidente que agrupe un buen número de ocurrencias de una clase. No obstante, esto no demuestra nada, pues tan solo hemos tomado 5 atributos de los 51 totales. Y además habría que sumarle las posibles combinaciones de estos.

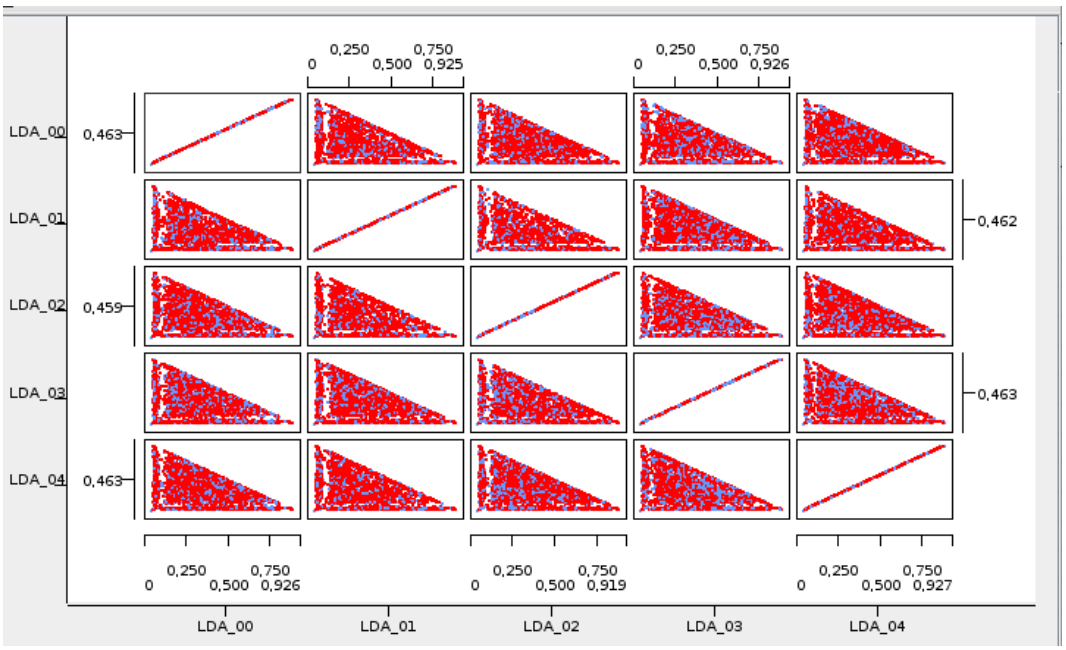

Figure 30: Distribución respecto LDA de 0 a 4.

A continuación, tenemos otra gráfica (figura 31) que enfrenta el porcentaje de palabras negativas de la noticia con el porcentaje e positivas. Como cabe de esperar, hay una dependencia lineal exacta. El problema reside, una vez más en que las ocurrencias de sendas clases están mezcladas.

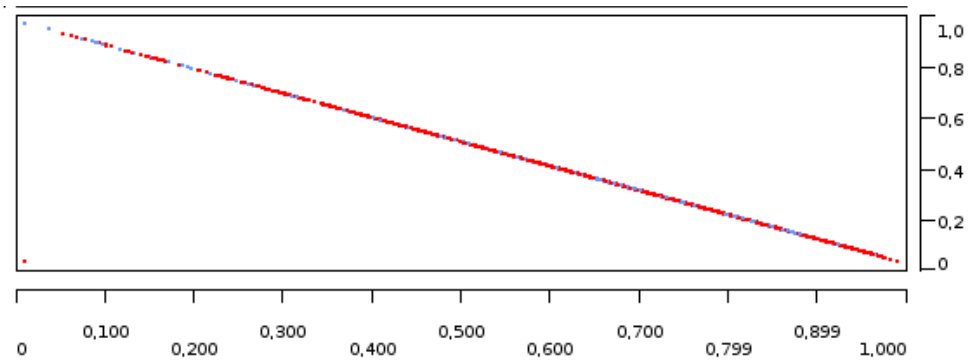

Figure 31: Dependencia lineal entre rate_negative_words y rate_positive_words.

Y una vez más con otros dos atributos en la figura **??**. En esta ocasión he querido reflejar también la probabilidad de popular que asigna el algoritmo KNN a cada ocurrencia. Cuanto más grande el punto, mayor es la probabilidad que otorga el KNN de que sea una noticia popular. El resultado bueno sería que los puntos de menor dimensión fueran rojos y los de grande azules, no es el caso.

De hecho, hay puntos grandes azules la misma situación que otros rojos y pequeños, y viceversa. En resumen, hay datos que por el resto de atributos se clasifican bien, pero en este atributo falla. Y al revés, otros atributos lo clasifican mal, pero en este atributo si parece estar bien clasificado.

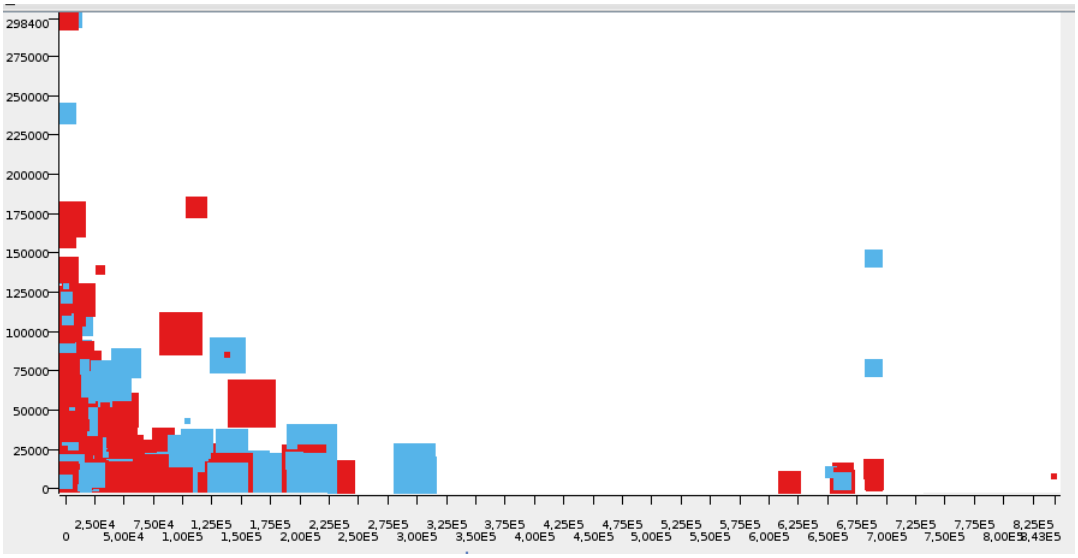

Figure 32: Inconsistencia para dos atributos.

Estos son tres ejemplos de posible inconsistencia que, realmente por sí solos, no demuestran que la haya. A pesar de esto, cada pareja de atributos que se compara parece tener las mismas consecuencias.

Para ilustrar finalmente este hecho y concluir la sección de análisis, vamos a ver el árbol de condiciones que produce el algoritmo C4.5.

Como ver el árbol entero es imposible debido a sus dimensiones, he tomado un conjunto de nodos hoja representativo. En los nodos hoja puede verse la proporción de noticias de cada clase. Notemos que el mínimo de instancias por hoja es 40, por tanto, cuándo un nodo alcanza menos de 80 instancias por hoja ya no se puede especializar más la rama.

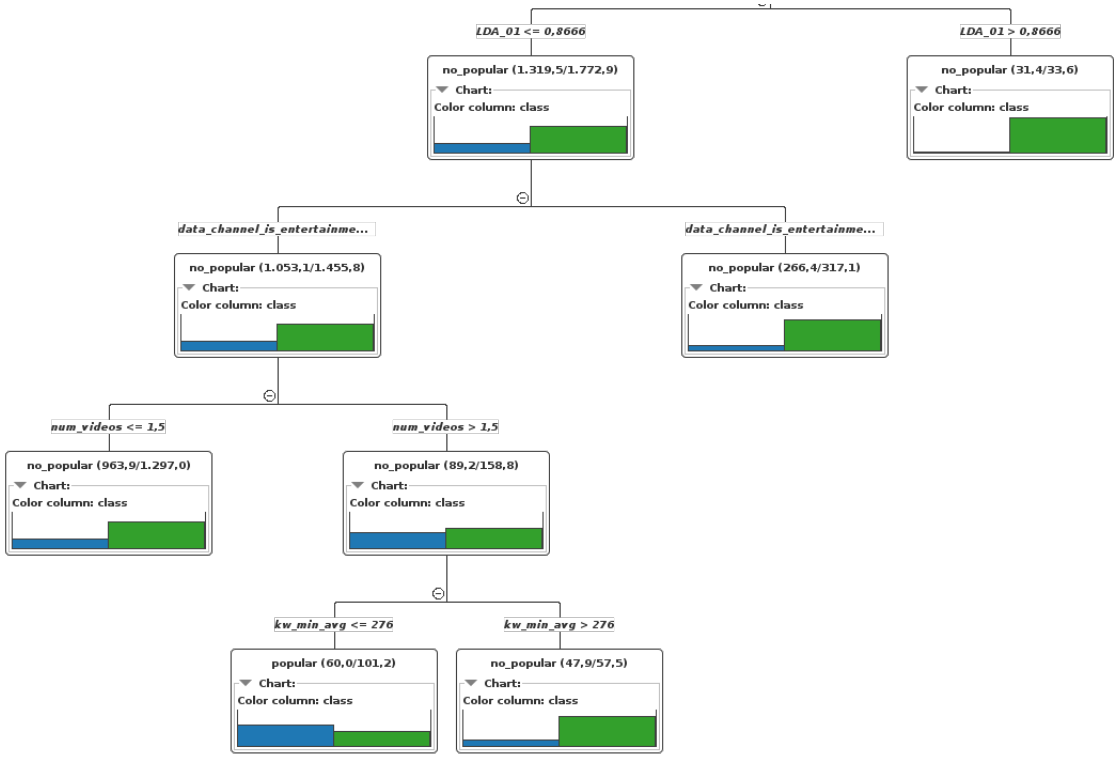

Figure 33: Hojas del árbol de decisión C4.5.

En la figura 33 puede verse como hay varios nodos que ya no pueden dividirse más pero que siguen manteniendo un cierto número de noticias de la clase contraria a la clasificada. Esto es, de 39644 noticias, no es capaz de separar menos de 80 con atributos similares de la misma clase. Una vez más, encontramos una razón para asegurar que el conjunto de datos muestra inconsistencias, quizás no pura inconsistencia pero cercana a esta.

7 Bibliografía

References

[1] Repositorio de datos en uci. 2015. (http://archive.ics.uci.edu/ml/datasets/Online+News+Popularity).