Visión por computador
Relación 1

Guillermo Gómez Trenado | 77820354-S
guillermogotre@correo.ugr.es

8 de noviembre de 2018

Índice

1. Introducción

En esta práctica el problema al que nos enfrentamos es la predicción de la popularidad de una noticia en base a sus características, para eso se ha definido el número de veces compartida una noticia para que sea exitosa como 3000, se han tabulado las características que describo en el siguiente apartado y se ha catalogado como "popular.º "no popular". La muestra se ha extraido de la web *Mashable.com*.

1.1. Propiedades de las instancias

Para clasificar cada instancia contamos con 58 propiedades más la clase[1], morfológicas, semánticas y contextuales. Por un lado contamos con características como el número de palabras en el título, subtítulo, texto y etiquetas, el número de enlaces, de imágenes o de videos, e incluso la longitud media de las palabras que aparecen en el texto. Por otro lado, mediante el uso de algoritmos de minería de opinión o análisis de sentimientos se ha evaluado el texto asignando un valor cuantitativo de polaridad a cada palabra y se define tanto mínimo, máximo y media para polaridad positiva y negativa como el porcentaje de palabras positivas y negativas respecto a la totalidad del texto; asimismo mediante *Latent Dirichlet Allocation*[2] se analizaron los artículos de toda la web definiendo las cinco categorías más populares y se define la distancia a la primera, a las dos primeras y así hasta la distancia hasta las cinco categorías más populares. Por último, contamos también con información relativa al contexto, como el día de la semana y la popularidad de las etiquetas del artículo. Esta no es una descripción exhaustiva[3] sino que el propósito de la descripción es dibujar de un brochazo el tipo de información con la que contamos para estimar la popularidad de una noticia.

1.2. Características de la muestra

Por la propia naturaleza de la muestra nos encontramos con dos clases **no balanceadas**, pues es mucho menos frecuente encontrar noticias populares que no populares, de ahí el interés que suscita este tivo de investigaciones, a fin de no sólo predecir la potencial popularidad de una noticia, sino el potencial derivado de este conocimiento para modificar las noticias a fin de estimular su popularidad.

Del mismo modo, tal como podemos ver en el apéndice 1, en la muestra aparecen hasta 379 celdas vacías por columna repartidas de forma heterogenea por los datos, que de los más de 30000 ejemplos que tenemos en la muestra de aprendizaje no debería suponer un obstáculo significativo pero sí que tendremos

[1] FERNANDES (2015)

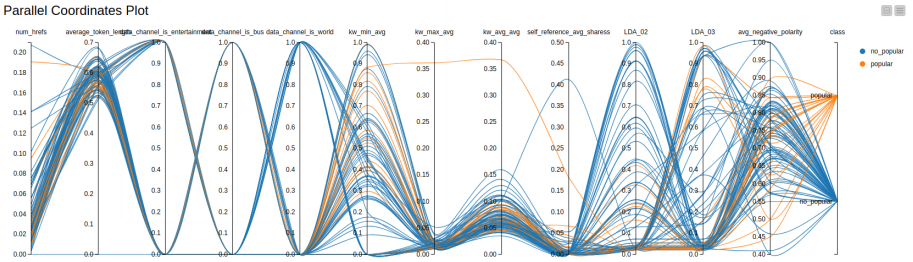
[2] BLEI (2003)

[3] Dataset, accedido el 4 de Noviembre `https://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/InteligenciaDeNegocio/Curso18-19/onlinenewspopularity.names`

que atajar este problema para algunos algoritmos de aprendizaje. Los datos no están normalizados, y encontramos tanto variables numéricas como categóricas.

En el apéndice 2 podemos ver la matriz de correlación. En primer lugar podemos apreciar cómo ninguna variable tiene una fuerte correlación con la clase de salida, yendo desde −,124 hasta 0,181, las más destacables por si inmediato sentido semántico son una distancia baja a las dos primeras categorías del *LDA* —sin embargo está inversamente relacionado la distancia a las tres primeras—, pertenecer al canal de *mundo* o *entreteniemiento*, o que la longitud media de las palabras sea baja.

Sin embargo, como vemos en la siguiente gráfica —hecha con las doce características con mayor valor absoluto de correlación—, el probelma no parece ser fácilmente separable, al menos de forma lineal.



2. Resultados Obtenidos

2.1. Transformaciones preliminares

El primer procesado de los datos que realizamos es la normalización a fin de facilitar el aprendizaje a algoritmos sensibles a esto como son el KNN —que la distancia euclídea estándar falla para evaluar la distancia entre dos instancias de forma homogénea para todas las características en datos no normalizados—, o las redes neuronales —que los datos no normalizados ralentizan la convergencia debido a la propagación hacia atrás de rangos muy dispares—.

Además de la anterior he aplicado otros dos preprocesados a la totalidad de los datos aunque se nos aconsejo en el sentido contrario, las razones para esto son dos, primero una teórica, el preprocesado por columnas analiza únicamente una variable en su totalidad, dándonos una información analítica no relacionada con la clase a estimar, sino de la naturaleza del tipo de dato, por lo tanto, de la misma forma que la normalización, este tipo de transformaciones no nos ofrece una ventaja poco realista sobre nuestra capacidad de clasificación; por otro lado, una razón práctica, a medida que vayamos aplicando otros algoritmos de preprocesado que sí tienen en cuenta la totalidad de las columnas, estos sí realizados dentro de la validación cruzada, tratar con un número excesivo de características ralentiza tanto el análisis que resulta impracticable, y al fin y al cabo, el objetivo de esta práctica no es obtener el mejor resultado posible sino dar nuestro primer chapoteo en la minería de datos, y un flujo de trabajo ágil

sirve infinitamente mejor a este propósito.

El primer preprocesado realizado es un **filtro de baja varianza**, como lo realizamos después de la normalización buscamos variables que sean próximas a la columna constante excepto en unos pocos valores, este es el primer sacrificio que hacemos, es posible —aunque ahora veremos que no parece probable— que estas variables sean tremendamente útiles para la clasificación, pero lo más habitual es que estas variables con baja varianza en el peor de los casos resten algo de información al algoritmo de aprendizaje, y en el mejor de los casos introduzcan una distracción a dichos algoritmos que nos interesa eliminar. Podemos observar en la siguiente table, que un umbral de 0,01 —de todos los valores con los que probé—, evaluado con Random Forest nos permite pasar de 52 columnas a 32, conservando casi la totalidad de nuestra capacidad predictiva a juzgar por la última columna del análisis que corresponde al área bajo la curva ROC devuelto por KNIME.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Area ...
RandomForest (Normalizer)	578	560	30158	8348	0.508	0.065	0.982	0.115	0.018	0.935	0.523	0.252	0.181	0.775	0.698
RandomForest (Normalizer + Variance)	387	437	30281	8539	0.47	0.043	0.986	0.079	0.014	0.957	0.515	0.207	0.143	0.774	0.674

El segundo preprocesado que aplicamos fuera de la validación cruzada es un **filtro de correlación**, aquí queremos eliminar la redundancia de datos, como dejamos fuera del análisis la correlación con la clase a predecir lo que medimos es la correlación entre variables explicativas y eliminar aquellas que se expliquen mutuamente, para quedarnos sólo con una de ellas. En este caso un umbral de 0,8 —de todos los valores con los que experimenté— nos permite conservar casi todo nuestro poder de clasificación en las mismas condiciones que en el párrafo anterior eliminando dos columnas más, es decir, pasando de 32 variables a 30.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Area ...
RandomForest (Normalizer + Variance)	387	437	30281	8539	0.47	0.043	0.986	0.079	0.014	0.957	0.515	0.207	0.143	0.774	0.674
Random Forest (Normalizer + Variance + Correlation)	411	449	30269	8515	0.478	0.046	0.985	0.084	0.015	0.954	0.516	0.213	0.148	0.774	0.674

Con todo lo anterior hemos conseguido reducir la dimensionalidad del problema conservando prácticamente la totalidad de la información a la luz de los resultados preliminares arrojado por el clasificador, he decidido utilizar Random Forest para el análisis pues es el que mejor resultados arrojaba en el *paper* original. Como ya comentaba, este tipo de preprocesado no creo que sea descabellado hacerlo fuera del proceso de entrenamiento y sobre la totalidad de los datos —como he visto que se hace habitualmente, incluso en la documentación de KNIME [4]—; permitiéndonos hacer un ulterior análisis más ágil e interesante.

2.2. Medidas de calidad y consideraciones afines

Para todos los experimentos que vamos a realizar de ahora en adelante —y los dos anteriores— vamos a tomar como clase positiva la clase "popular", y las medidas de calidad reflejarán este hecho y no el contrario —clasificación binaria— "no popular". El motivo último de esta decisión radica en la utilidad última de este tipo de predictores, queremos ser capaces de estimar cuándo un artículo particular será popular por el impacto económico que tiene esta circunstancias para empresas de noticias online que aspiran a la viralización de sus entradas logrando la máxima difusión posible.

[4] Documentación de KNIME, accedido el 4 de Noviembre `https://www.knime.com/nodeguide/applications/model-selection-and-management/model-selection-sampled`

Lo anterior tiene repercusión en varios aspectos, el primero es, qué medidas de las que evaluaremos son las más interesantes para nuestro problema, pues esto varia ampliamente dependiendo del problema, de forma análoga a como sucede en una cadena de montaje, si producimos piezas de valor ínfimo lo más probable es que nos interese hacer tantas como podamos a la mayor velocidad posible descartando posteriormente aquellas defectuosas, sin embargo, si lo que producimos son por ejemplo vehículos, o equipos de alta precisión, lo que nos interesa es no tanto sacrificar la eficacia sino la eficiencia, reduciendo la velocidad de producción pero garantizando una tasa de éxito —producto sin defectos— lo más alta posible, por los altos costes asociados a un fallo. En este problema que tenemos entre manos nos encontramos con una circunstancia análoga a la descrita en primer lugar, queremos saber qué noticias serán exitosas con dos objetivos, el primero probablemente invertir en mayor publicidad para éstas y estimular la visibilidad de las mismas para el público; y en segundo lugar buscar características que parezcan indicar el potencial de popularidad con el objetivo último de modificar las propias noticias formalmente para aumentar la visibilidad general del portal de noticias. Por lo anterior nos interesarán más aquellos algoritmos de aprendizaje cuyas medidas nos indiquen no sólo su alta capacidad predictiva sobre el conjunto de evaluación, sino en qué medida estos favorecen la predicción de la clase positiva, aun a riesgo de clasificar erróneamente artículos *no populares* como *populares*.

Contamos con distintas medidas para evaluar el rendimiento, además de las cuatro

1. Los cuatro valores de la matriz de confusión —*true positives*, *false positives*, *true negatives* y *false negatives*—, tomando como clase positiva la clase *popular*.

2. *Positive predictive value*, que nos indica la proporción de aciertos entre todas las instancias clasificadas positivamente.

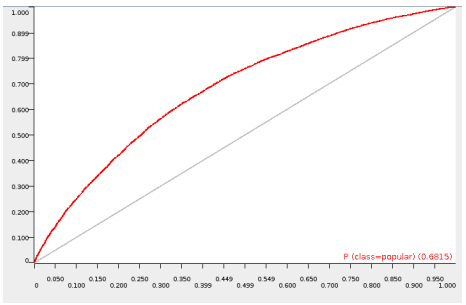
3. *True positive rate* y *True negative rate* que nos indican la proporción de aciertos respecto al *ground truth* para la clase positiva y negativa.

4. *F1-score*, que relaciona la proporción de aciertos entre las clasificadas positivamente y la proporción de aciertos respecto a la verdad. En definitiva, este valor será alto cuando ambas medidas sean altas y bajo cuando uno de los dos —o los dos— sean bajos.

$$F1 = 2\frac{PPV \cdot TPR}{PPV + TPR}$$

5. *False positive rate* y *False negative rate*, que nos indican qué proporción hemos clasificado incorrectamente respecto al total de la clase que debían clasificar, intuitivamente esto es qué proporción de instancias estamos dejando de clasificar correctamente.

6. *Area bajo la curva ROC*. La curva ROC dibuja la relación entre el TPR y el FPR, desplazando para esto el umbral de corte en la confianza de la predicción para clasificar positiva o negativamente. El área bajo la curva es una medida entre 0 y 1 que define qué proporción de la gráfica está bajo esta curva, el peor caso posible es 0,5 —recta diagonal— pues en clasificación binaria, clasificar correctamente la mitad de los casos es equivalente a tirar una moneda al aire y apuntar las caras, tanto un valor próximo a 0 como un valor próximo a 1 serían clasificadores deseables, el segundo de forma evidente, y el primero porque un clasificador que falle sistemáticamente es tan bueno como el que acierte siempre, sólo hay que invertir la predicción.



7. *G-mean*, medida de equilibrio entre los aciertos para ambas clases.

$$Gmean = \sqrt{TPR \cdot TNR}$$

8. *G-measure*, relaciona los aciertos respecto a los clasificados positivamente y los de la clase positiva. Si definiéramos un clasificador que siempre estima la clase positiva, su TPR sería igual a 1, pero su PPV sería igual a la proporción de instancias pertenecientes a la clase positiva. Esta medida nos da la media geométrica de ambas medidas.

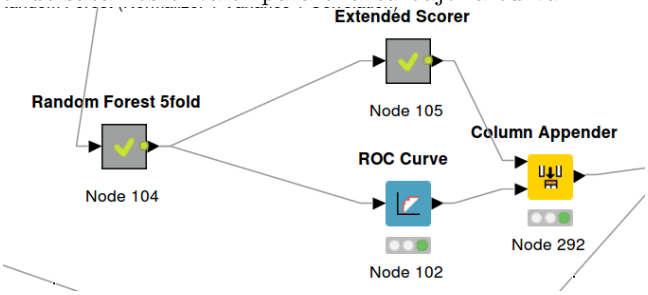
$$Gmeasure = \sqrt{PPV \cdot TPR}$$

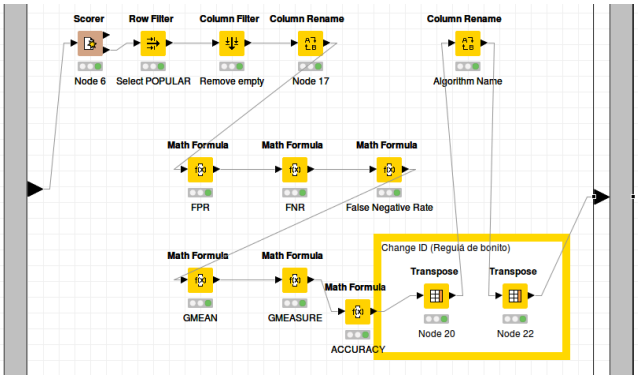
9. *Accuracy*, proporción de aciertos entre la totalidad de los ejemplos.

No podemos llevar a cabo nuestro análisis basándonos únicamente en una de estas medidas, en primer lugar por la parcialidad de cada una de estas, y en segundo lugar, por las circunstancias a las que hacíamos referencia en la analogía de la cadena de montaje. En nuestro análisis nos centraremos como medida de calidad e intentamos maximizar tanto el bajo la curva ROC —que es buen indicador de la calidad de un predictor, aunque es susceptible al desplazamiento del umbral de corte mientras que nosotros siempre separamos las dos clases en una confianza de 0,5—; y $F1 - score$, pues de alguna manera nos orienta sobre nuestra capacidad predictiva de la clase positiva que es la que nos interesa especialmente. El valor de *accuracy* en nuestro caso es poco significativo, pues

al estar la clase profundamente desbalanceada el algoritmo que clasifique con la clase mayoritaria, como veremos a continuación obtendría un valor de 0,775, el cual podría parecer a primer vista razonablemente bueno, y sin embargo expresa una nula capacidad predictiva, más aún para nuestro objetivo de priorizar las clasificaciones positivas.

A cada salida de los resultados de un modelo de aprendizaje le aplicamos dos nodos, *Extended Scorer*, donde calculamos las métricas descritas, y *ROC Curve*, de donde sacamos el valor para el area bajo la curva.





3. Análisis de resultados

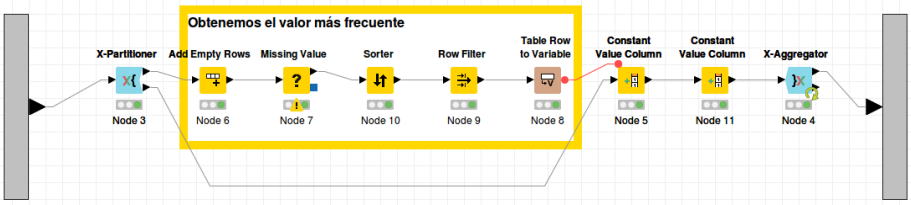
Hemos tomado 8 algoritmos de aprendizaje de los que posteriormente conservaremos 6: ONER, Naives Bayes, MLP, C4.5, Gradient Boosted, Random Forest, Adaboost y KNN. En los siguientes epígrafes los describiremos individualmente, sin embargo, al respecto del conjunto de clasificadores puedo decir que he intentado obtener una muestra de familias heterogénea, con prevalencia de aquellos basados en árboles sencillamente porque suelen ser algoritmos rápidos y lamentablemente KNIME no es el entorno más eficiente para problemas grandes y el interés último de esta práctica es experimentar con un problema real con dificultades reales, la definición de hiperparámetros de los clasificadores y el preprocesado de datos a fin de favorecer el aprendizaje. Y esto sería impracticable con ejecuciones individuales de 10 o 15 minutos.

Describiremos cada algoritmo y al final veremos la comparativa de resultados.

Tanto para este apartado como para el resto he decidido no mostrar la gráfica de la curva ROC pues si bien es una herramienta muy interesante para medir la bondad de un modelo, la comparativa entre ellas resulta pobre y torpe, y utilizaremos las métricas antes mencionadas para enfrentar los algoritmos y reflexionar sobre sus resultados.

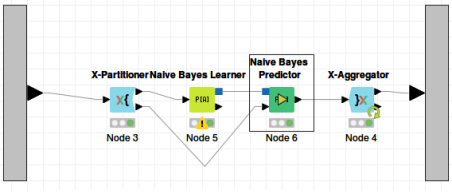
3.1. ONER

Este es el algoritmo de referencia, y la métrica de cualquier otro algoritmo de aprendizaje debería superar al menos éste. Analizamos la clase más frecuente y predecimos ese valor con confianza igual a 1.



3.2. Naives Bayes

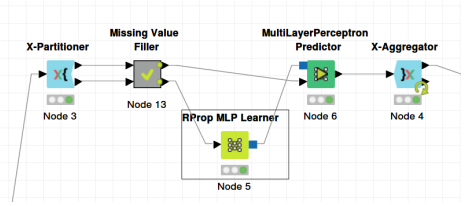
Es un clasificador probabilístico, basado en el teorema de Bayes, que calcula la probabilidad de obtener la clase positiva condicionada a las condiciones de la instancia. Para poder realizar el cálculo se asumen que las características son independientes. Como utiliza la frecuencia, o una función de densidad —variables categóricas o numéricas respectivamente— para el cálculo de la probabilidad es vulnerable a muestras desbalanceadas como es nuestro caso.



3.3. Perceptrón Multicapa

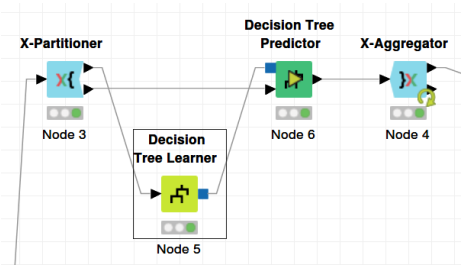
Este algoritmo es un regresor, como todos los regresores se puede adaptar a un problema de clasificación pero no es siempre posible en el sentido contrario. Utiliza una colección de perceptrones distribuidos en capas —combinación lineal con los valores de entrada— con una función no lineal entre capa y capa —normalmente reLU— para el cálculo de los valores de salida. En el modelo de clasificación binaria al valor de salida le aplica la función sigmoide para obtener un valor entre 0 y 1 que estime la probabilidad de pertenecer a una u otra clase. Para cada error va actualizando los pesos de cada nodo con la derivada parcial evaluada en ese nodo multiplicado por el error y una tasa de aprendizaje. Idealmente este modelo es robusto contra clases desbalanceadas pues cuando el

error es igual a 0 el valor de propragación es 0. Para este algoritmo he tenido que aplicar preprocesado pues no puede trabajar con celdas vacías, en el apartado correspondiente hablaremos de la técnica de *data imputation* utilizada, se resume en una primera imputación con un regresor lineal, y una segunda imputación para los valores aún ausente debido a problemas en el cálculo por otras celdas vacías donde aplico el valor medio.



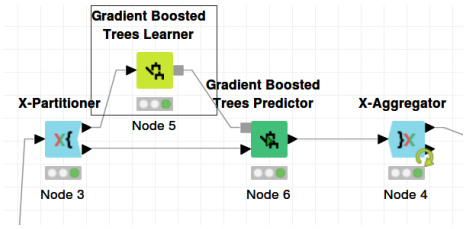
3.4. C4.5

Este es uno de los algoritmos basados en árboles de decisión que vamos a utilizar, aunque es superado habitualmente en la literatura por otros algoritmos más robustos, es interesante tenerlo aquí porque es un algoritmo rápido que será la base de otros nodos que usaremos como en la implementación posterior que haremos del CVCF. Tolera celdas vacía y evita el sobreaprendizaje con estrategias de poda.



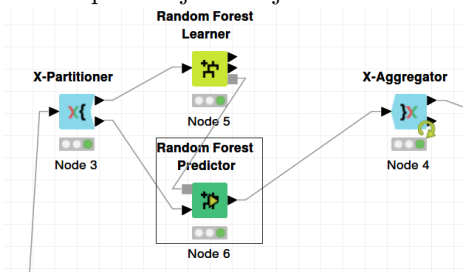
3.5. Gradient Boosted Classifier

Utiliza una colección de predictores débiles obtenidos de forma iterativa —*boosting*— formados por árboles de poca profundidad en nuestro caso y los optimiza posteriormente con el gradiente de una función de pérdida diferenciable al igual que hace el perceptrón multicapa. Permite realizar la modificación del algoritmo con gradiente estocástico tomando muestras aleatorias de la totalidad de las instancias, en este apartado usamos la configuración por defecto que utiliza la totalidad de los datos en cada aplicación del gradiente descendiente. Debido a la optimización mediante el gradiente es robusto contra clases desbalanceadas pero puede adolecer de sobreajuste. He elegido este método porque posee la potencia de los *ensamble learners* y de los modelos optimizados por escalada y parece un candidato competente.



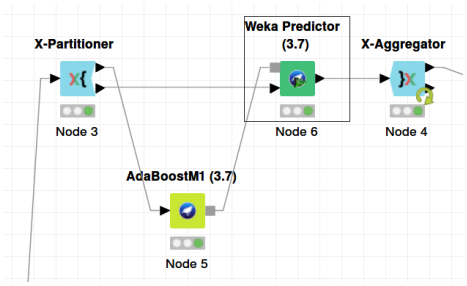
3.6. Random Forest

Este algoritmo, al igual que el anterior se basa en una colección de clasificadores débiles basados en árboles de decisión aunque en vez de ir aprendiendo el error de otro clasificador débil, entrena numerosos árboles pequeños en una porción de las características de entrada, utilizando posteriormente un proceso de votación para definir la clase de salida. Por la forma en la que se define la ganancia para generar los nodos de los árboles este tipo de modelos es vulnerable a conjuntos de datos no balanceados. He elegido este algoritmo por dos motivos, dentro de la familia de *ensamble learners* utiliza la técnica de *bagging* como ya hemos descrito en vez de *boosting* como el anterior, y además es referenciado en el *dataset* como el algoritmo que arroja el mejor resultado.



3.7. Adaboost

Este otro método basado en *boosting* con árboles de decisión, la diferencia con *Gradient Boosted Classifier* es que no utiliza gradiente descendente para optimizar los árboles y además va adaptando los árboles ulteriores a las características de las muestras mal clasificadas por los árboles sencillos anteriores. Lo he elegido porque también viene referenciado en el *dataset* y puede ser interesante compararlo con el clasificador *Gradient Boosted*. Es, al igual que el anterior vulnerable al ruido y los *outliers*, así como al sobreaprendizaje, aunque puede ser paliado limitando el número de árboles.



3.8. KNN

Es el último clasificador del conjunto, si bien no es viable en entornos de producción, por su relevancia histórica y por sus especiales características —los datos son el modelo— es siempre interesante tomarlo de referencia en cualquier problema de aprendizaje. Los problemas de sobreaprendizaje se pueden paliar hasta cierto punto definiendo un número de vecinos grande, aunque esto penaliza el resultado de las estrategias para evitar la vulnerabilidad en problemas no balanceados, que consisten en ponderar el voto de cada vecino por la distancia hasta la instancia evaluada. Como no puede calcular la distancia sobre instancias con celdas vacías ignora éstas; este problema lo atajaremos en el apartado del preprocesado.

3.9. Resultados preliminares

Utilizando la configuración por defecto en cada algoritmo obtenemos los siguientes resultados.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Area...
Gradient Boosted	348	395	30323	8578	0.468	0.039	0.987	0.072	0.013	0.961	0.513	0.196	0.135	0.774	0.684
Random Forest	416	429	30289	8510	0.492	0.047	0.986	0.085	0.014	0.953	0.516	0.214	0.151	0.775	0.673
MLP (Missing: Linear Interpol/MFV + Mean)	345	346	30372	8581	0.499	0.039	0.989	0.072	0.011	0.961	0.514	0.195	0.139	0.775	0.666
Adaboost	27	28	30690	8899	0.491	0.003	0.999	0.006	0.001	0.997	0.501	0.055	0.039	0.775	0.64
Naives Bayes	4311	9205	21513	4615	0.319	0.483	0.7	0.384	0.3	0.517	0.592	0.582	0.392	0.651	0.633
C4.5	978	1532	29186	7948	0.39	0.11	0.95	0.171	0.05	0.89	0.53	0.323	0.207	0.761	0.6
kNN	1704	3369	23400	6059	0.336	0.22	0.874	0.266	0.126	0.78	0.547	0.438	0.272	0.727	0.563
ONER	0	0	30718	8926	?	0	1	?	0	1	0.5	0	?	0.775	0.5

Antes de realizar el análisis hay que aclarar que tenemos dos medidas para el area bajo la curva, la primera es una aproximación para clasificadores que no devuelven medida de confianza, mientras que la segunda sí está calculada a través de aproximaciones a la integral, es ésta segunda la que utilizamos y dejamos la primera como referencia.

Vemos que los tres algoritmos que mejor desempeño tienen según esta medida son *Gradient Boosted*, *Random Forest* y MLP —habrá que ver si MLP sigue siendo competitivo cuando todos los demás algoritmos tengan imputación de datos aplicada—. Vemos para el AUC valores superiores a 0,5 lo cual siempre es una buena noticia, pero sin embargo vemos que la otra medida que nos interesa, $F1$, tiene valores por debajo de 0,1, esto se debe al bajo TPR; los algoritmos que mejor lo area obtienen no coinciden con el que posee mejor valor para $F1$ —Naives Bayes—, esto se debe sin duda alguna a la muestra no balanceada, que está premiando la selección de la clase dominante —casi 4 veces mayor—, esto no es sorpresa para *Gradient Boosted* y *Random Forest*,

pero sí que lo es para MLP, el motivo de esta vulnerabilidad a la falta de balanceo sea posiblemente un número reducido de iteraciones para el gradiente descendiente, lo que dirigirá los pesos de los nodos en un primer momento a producir como resultado valores que favorezcan a la clase dominante, es posible que si dejáramos más iteraciones el algoritmo corriendo converja a soluciones que hagan subir el valor para $F1$, aunque haría seguramente falta ir enfriando la tasa de aprendizaje progresivamente para evitar que la medida de error vaya zigzagueando, lamentablemente en KNIME no podemos hacer este tipo de análisis aunque probablemente tengamos oportunidad de analizarlo en prácticas posteriores.

4. Configuración de algoritmos

A partir de este apartado nos vamos a quedar con sólo seis algoritmos de aprendizaje, descartando el *ONER* y *Naives Bayes*, aunque tendremos ocasión de probarlo luego cuando balanceemos las clases.

Aunque lo interesante sería definir las horquillas de los valores para cada uno de los n parámetros que queremos modificar e ir evaluando el algoritmo en muestras aleatorias de ese volumen de n dimensiones definido por el espacio de parámetros posibles, esto no es tan fácil con KNIME como lo puede ser en Python —habría que generar todas las tablas, ir sacando los valores e introducirlos al algoritmo mediante variables de flujo—, así que lo que vamos a hacer es evaluar para cada clasificador 5 configuraciones distintas evaluando hipótesis que nos interesan.

Este análisis ofrece una dificultad añadida, y es que en la estimación de la potencia de un algoritmo en base a métricas dispares no hay un procedimiento definido, y esta tarea se hace harto más complicada cuando sin métodos estadísticos no podemos afirmar cuando una diferencia es estadísticamente significativa, especialmente para diferencias entre 0,05 y 0,005.

4.1. Perceptrón Multicapa

Vamos a probar dos hipótesis, en primer lugar comprobar si tal y como especulábamos, el bajo valor de $F1$ se debe al bajo número de *epochs*; y en segundo lugar, ver el impacto de ampliar la profundidad del árbol con el mismo número total de nodos ocultos. La configuración por defecto utiliza 2 capas ocultas de 10 neuronas cada una y un total de 100 iteraciones. Es sorprendente que el algoritmo de MLP implementado en KNIME no permita modificar la tasa de aprendizaje, al menos de forma inmediata a través del panel de configuración.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Area...
MLP 500 iters	731	868	29850	8195	0.457	0.082	0.972	0.139	0.028	0.918	0.527	0.282	0.193	0.771	0.663
MLP 200 iters	643	695	30023	8283	0.481	0.072	0.977	0.125	0.023	0.928	0.525	0.265	0.186	0.774	0.669
MLP 1x20	421	454	30264	8505	0.481	0.047	0.985	0.086	0.015	0.953	0.516	0.216	0.151	0.774	0.669
MLP (Missing: Linear Interpol/MFV + Mean)	345	346	30372	8581	0.499	0.039	0.989	0.072	0.011	0.961	0.514	0.195	0.139	0.775	0.666
MLP 3x7	195	192	30526	8731	0.504	0.022	0.994	0.042	0.006	0.978	0.508	0.147	0.105	0.775	0.666

Al respecto del primer punto podemos ver que efectivamente parece que los pobres resultados iniciales se debían a un bajo tiempo de aprendizaje, y al aumentar el número de iteraciones conseguimos resistencia frente al desabalanceo

de la muestra. Como era de esperar la relación no es lineal y es que a medida que el MLP se acerca al punto de convergencia cada vez es más difícil aprender —imposible con la tasa de aprendizaje demasiado alta—. El tiempo de ejecución del MLP con 500 iteraciones es demasiado alto, pero vamos a utilizar a partir de ahora el modelo con una sola capa oculta de 20 nodos y 200 iteraciones.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Area ...
MLP ...	648	666	30052	8278	0.493	0.073	0.978	0.127	0.022	0.927	0.525	0.267	0.189	0.774	0.673

Es un fenómeno estudiado en la literatura que aumentar el número de capas permite reducir el número de nodos para calcular cualquier función —reduciendo la complejidad en espacio del modelo— pero dificulta el aprendizaje porque el valor de *backpropagation* en las capaz inferiores va acercándose cada vez más a 0, y hay que utilizar estrategias —no implementadas en KNIME— que favorezcan el aprendizaje en estas capas, como el modelo residual[5] o *highway*[6]. En nuestro caso observamos cómo la profundidad del modelo está inversamente relacionada con la calidad de los resultados, es posible que pudieramos paliar parcialmente este fenómeno con un número mayor de iteraciones pero no nos interesa en este punto de la práctica.

4.2. C4.5

Para este algoritmo en primer lugar analizamos las dos métricas estudiadas en clase, *GainRatio* y *GINI* —por defecto—; y en segundo lugar estudiamos el efecto de la morfología del arbol: con y sin poda; y definiendo el número mínimo de nodos por hoja a 2 —por defecto—, 4 y 8.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
C4.5 min 8	902	1460	29258	8024	0.382	0.101	0.952	0.16	0.048	0.899	0.527	0.31	0.196	0.761	0.607
C4.5 min 4	875	1404	29314	8051	0.384	0.098	0.954	0.156	0.046	0.902	0.526	0.306	0.194	0.762	0.607
C4.5	978	1532	29186	7948	0.39	0.11	0.95	0.171	0.05	0.89	0.53	0.323	0.207	0.761	0.6
C4.5 gain ratio	28	34	30684	8898	0.452	0.003	0.999	0.006	0.001	0.997	0.501	0.056	0.038	0.775	0.567
C4.5 no pruning	2265	5452	25266	6661	0.294	0.254	0.823	0.272	0.177	0.746	0.538	0.457	0.273	0.694	0.54

Se aprecia que el desempeño del algoritmo con la métrica de ramificación *GainRatio* es peor que *GINI*. Asimismo se observa que la ausencia de poda, consigue una valor *F1* mayor pero disminuye significativamente el área, esto se debe seguramente a que la falta de poda nos devuelve un clasificador parecido al 1NN, y de hecho la proporción de clasificados positivamente y los clasificados negativamente refleja un poco la proporción de la muestra *populares/no populares*. En el momento que empezamos a podar y a aumentar el número mínimo de ejemplos por nodo nos encontramos con que el efecto de la falta de balanceo aumenta, quedando probablemente perdidos los pocos valores positivos entre la generalización producida por la poda, donde la clasa mayoritara domina al clasificador. Parece que el C4.5 por defecto, con métrica GINI y poda con mínimo 2 nodos por hoja supone el mejor equilibrio entre la capacidad de predicción sobre la muestra y la capacidad de predicción de los casos positivos, éste será el que utilizaremos.

[5] https://arxiv.org/abs/1512.03385
[6] https://arxiv.org/abs/1505.00387

4.3. Gradient Boosted Classifier

Para este algoritmo nos interesa analizar el efecto en el resultado de varios parámetros: limitar el número máximo de niveles de un árbol a 2 —en vez de 4 por defecto— para conseguir árboles verdaderamente superficiales y a 8, para conseguir árboles de mayor profundidad que consigan mayor separación en cada árbol del *boosting*; después hacer sampleo sobre las n variables, tomando $sqrt(n)$ variables en cada árbol —y aumentando de 100 árboles por defecto a 600— para conservar el mismo número de variables usadas en total; aumentar al doble —200 frente a 100— el número de árboles, y por último aplicar sampleo sobre las instancias.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
Gradient Boosted	348	395	30323	8578	0.468	0.039	0.987	0.072	0.013	0.961	0.513	0.196	0.135	0.774	0.684
Gradient Boosted 200 + 0.5	630	672	30046	8296	0.484	0.071	0.978	0.123	0.022	0.929	0.524	0.263	0.185	0.774	0.683
Gradient Boosted 200 trees	542	548	30170	8384	0.497	0.061	0.982	0.108	0.018	0.939	0.521	0.244	0.174	0.775	0.683
Gradient Boosted sqrt + 600	656	770	29948	8270	0.46	0.073	0.975	0.127	0.025	0.927	0.524	0.268	0.184	0.772	0.678
Gradient Boosted min 8	803	979	29739	8123	0.451	0.09	0.968	0.15	0.032	0.91	0.529	0.295	0.201	0.77	0.676
Gradient Boosted 2 levels	97	105	30613	8829	0.48	0.011	0.997	0.021	0.003	0.989	0.504	0.104	0.072	0.775	0.674

Podemos apreciar cómo aumentar el número de árboles parece mejorar el valor de $F1$ sin deteriorar significativamente el valor del AUC. Por otro lado parece que 4 niveles por árbol es el punto dulce de equilibrio entre superficialidad de los árboles y demasiada especialización, el mejor valor para este parámetro suele ser estrictamente experimental y es difícil encontrar una justificación teórica pues dependerá del tipo de muestra. Por otro lado, utilizar árboles con conocimiento parcial —muestreo en las columnas— parece mejorar el $F1$ pero no podemos discriminar si es por el mayor número de árboles o por la eliminación del *bias*, sin embargo parece proporcionar un rendimiento ligeramente peor en el área. Parece entonces, que entrenar cada árbol con un muestreo aleatorio sin repetición de la mitad de la población y aumentar por 2 el número de árboles para compensar el número de filas total utilizadas en el algoritmo nos proporciona un equilibrio entre capacidad de predicción sobre la muestra total y potencia en la detección de casos positivos, ésta será la configuración que utilicemos.

4.4. Random Forest

En este caso, nos enfrentamos a algoritmo *ensamble* basado en *bagging* y los efectos de manipular el número mínimo de instancias por hoja y el número máximo de niveles por árbol deberá tener un efecto contrario.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
Random Forest 200	337	353	30365	8589	0.488	0.038	0.989	0.07	0.011	0.962	0.513	0.193	0.136	0.774	0.679
Random Forest min 2	239	221	30497	8687	0.52	0.027	0.993	0.051	0.007	0.973	0.51	0.163	0.118	0.775	0.673
Random Forest	416	429	30289	8510	0.492	0.047	0.986	0.085	0.014	0.953	0.516	0.214	0.151	0.775	0.673
Random Forest 10/2 200	0	0	30718	8926	?	0	1	?	0	1	0.5	0	?	0.775	0.624
Random Forest max 10	0	0	30718	8926	?	0	1	?	0	1	0.5	0	?	0.775	0.616
Random Forest max 5	0	0	30718	8926	?	0	1	?	0	1	0.5	0	?	0.775	0.57

Lo que observamos es interesante, y es que limitar la profundidad del árbol tiene un efecto perjudicial para el aprendizaje, pues por la falta de balanceo en la muestra, la generalización que se produce por cortar los árboles demasiado pronto siempre es hacia la clase dominante, y de hecho, mientras menor sea el valor máximo para la altura del árbol, peor es el rendimiento en AUC, con total seguridad debido a que la confianza que devuelve es aún mayor para la clase dominante. Vemos que todos los árboles con profundidad limitada han devuelto 0 instancias catalogadas con la clase positiva. Por otro lado, definir en

2 el número mínimo de instancias por hoja parece no afecta a la medida del área pero sí penaliza en la medida de $F1$, debido probablemente a la falta de balanceo donde la generalización favorece a la clase mayoritaria. Como la potencia de este algoritmo es el sampleo en columnas, parece que los mejores parámetros son los que venían por defecto, no poniendo límite al número mínimo de nodos por hoja, y no obligando a truncar los árboles a determinada altura.

Por último parece que aumentar el número de árboles beneficia de forma general al algoritmo, y es posible que esta sea una modificación trivial, si el problema puede efectivamente aprenderse, parece que introducir mayor variabilidad en las respuestas debido al incremento en el número de árboles puede producir un consenso más robusto, a costa de doblar el tiempo de cómputo.

4.5. Adaboost

Adaboost en Weka por defecto utiliza *Decision Stump*, vamos a probar el resultado con este algoritmo basado en árboles y con *C4.5*. Asimismo modificaremos el número de iteraciones de 10 —por defecto— a 50, y experimentaremos pasando el umbral de peso de 100 —por defecto— a 90, que comentan en la documentación puede acelerar el proceso de cómputo.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
Adaboost DS 50	149	218	30500	8777	0.406	0.017	0.993	0.032	0.007	0.983	0.505	0.129	0.082	0.773	0.662
Adaboost	27	28	30690	8899	0.491	0.003	0.999	0.006	0.001	0.997	0.501	0.055	0.039	0.775	0.64
Adaboost c45 50	1215	1842	28876	7711	0.397	0.136	0.94	0.203	0.06	0.864	0.538	0.358	0.233	0.759	0.636
Adaboost c4.5	1857	3868	26850	7069	0.324	0.208	0.874	0.253	0.126	0.792	0.541	0.426	0.26	0.724	0.599
Adaboost c45 weigth 90	2022	4435	26283	6904	0.313	0.227	0.856	0.263	0.144	0.773	0.541	0.44	0.266	0.714	0.595

En primer lugar, de manera informal, no he apreciado que reducir a 90 el valor para *weigth threshold* tenga un impacto significativo en el tiempo de ejecución. A juzgar por el área bajo la curva ROC, *DS* es un mejor algoritmo que *C4.5*, aunque el desempeño en la métrica $F1$ es significativamente peor. Vamos a utilizar el DS en Adaboost e intentaremos paliar posteriormente este efecto balanceando la muestra. Ésta es probablemente la comparación más difícil de este epígrafe pues no puedo posicionarme con seguridad en qué algoritmo puede ser más interesante para nuestro problema, más aún cuando queda todo el preprocesado pendiente.

4.6. KNN

Finalmente en el KNN las opciones de configuración son menores —ignorando la posibilidad de customizar la métrica de distancia—, modificaremos el número de vecinos —3, 5 y 9—, y el cálculo de la clase ponderado por la distancia de cada vecino; la fila no comentada utiliza 3 vecinos y peso en el cálculo de la clase.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
KNN 9W	1017	1530	25239	6746	0.399	0.131	0.943	0.197	0.057	0.869	0.537	0.351	0.229	0.76	0.595
KNN 5W	1366	2426	24343	6397	0.36	0.176	0.909	0.236	0.091	0.824	0.543	0.4	0.252	0.744	0.581
KNN 5NW	1364	2395	24374	6399	0.363	0.176	0.911	0.237	0.089	0.824	0.543	0.4	0.253	0.745	0.581
KNN 3 NW	1717	3397	23372	6046	0.336	0.221	0.873	0.267	0.127	0.779	0.547	0.439	0.273	0.727	0.565
KNN	1704	3369	23400	6059	0.336	0.22	0.874	0.266	0.126	0.78	0.547	0.438	0.272	0.727	0.563

Los resultados muestran que aumentar el tamaño de la vecindad mejora el resultado en AUC, pero esto se debe sin lugar a dudas por la falta de balanceo de la muestra, pues KNN es tremendamente vulnerable a este fenómeno, sin embargo, como posteriormente aplicaremos preprocesado para eliminar el ruido

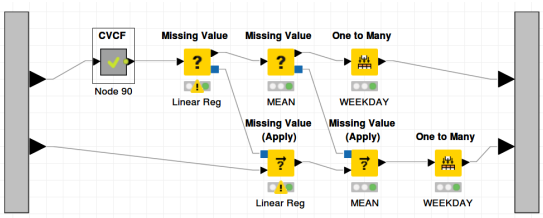
y balancear las clases vamos a quedarnos con la opción inicial, 3 vecinos ponderados por distancia, que ofrece el mejor valor para la medida $F1$.

5. Procesado de datos

Para el preprocesado de datos, además de normalizar y la reducción de carasterísticas que comentábamos en el apartado 2.1, realizo reducción de ruido e imputación de datos.

5.1. Imputación de datos y CVCF

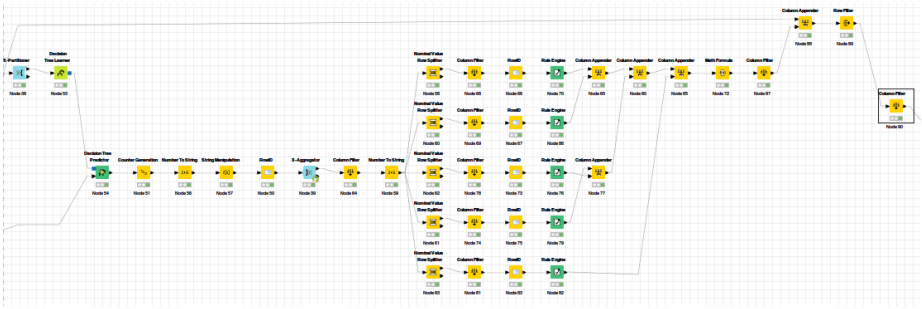
5.1.1. Imputación de datos



Por simplicidad analizamos este paso primero, aunque lo realizamos después de la reducción de ruido. Utilizamos el nodo *Missing Value* para imputar datos, hacemos una primera pasado utilizanod un regresor lineal, y como este método de imputación no puede resolver todas las celdas por dependencias con otras celdas vacías, añadimos un segundo paso en el que imputamos la media, esta decisión de usar la media se basa en el hecho de que aunque cambie la distribución de los datos reduciendo la desviación típica creo que representa mejor la naturaleza del dato, y en análisis preliminares que realicé se demostró que era más efectivo. Las variables categóricas utilizan el valor más frecuente para imputar el valor, me habría gustado probar con KNN para imputación pero KNIME no permite este algoritmo y no era inmediata su implementación. Finalmente transformo las variables categóricas en numéricas con *One to many* para homogeneizar la entrada de todos los clasificadores.

5.1.2. Cross-Validated Committees Filter (CVCF)

Como no era excesivamente complejo de implementar en KNIME, adapté el algoritmo al modelo de flujos con el que trabaja KNIME. En cada iteración de CV-5 aplico el modelo sobre la totalidad de los datos, aquellas instancias que no hayan podido ser clasificadas correctamente ni una sola vez aun habiendo sido simultaneamente parte del entrenamiento de C4.5 y de la evaluación son eliminadas en la salida.



Evidentemente, este algoritmo se aplica únicamente sobre la porción de entrenamiento del CV-5 exterior que evalúa el rendimiento de un algoritmo de aprendizaje cualquiera. Con este algoritmo consigo eliminar aproximadamente unas 200 instancias ruidosas en cada pasada por el nodo.

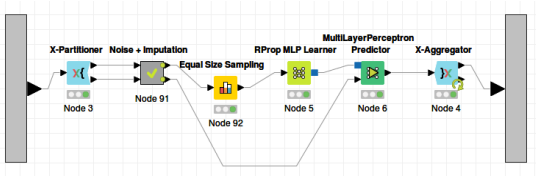
5.1.3. Resultados en los algoritmos

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
Gradient Boosted [Noise + Imputation]	574	676	30042	8352	0.459	0.064	0.978	0.113	0.022	0.936	0.521	0.251	0.172	0.772	0.685
Gradient Boosted	348	395	30323	8578	0.468	0.039	0.987	0.072	0.013	0.961	0.513	0.196	0.135	0.774	0.684
Random Forest [Noise + Imputation]	337	399	30319	8589	0.458	0.038	0.987	0.07	0.013	0.962	0.512	0.193	0.131	0.773	0.674
Random Forest	416	429	30289	8510	0.492	0.047	0.986	0.085	0.014	0.953	0.516	0.214	0.151	0.775	0.673
MLP [Noise + Imputation]	385	409	30309	8541	0.485	0.043	0.987	0.079	0.013	0.957	0.515	0.206	0.145	0.774	0.667
MLP (Missing: Linear Interpol/MFV + Mean)	345	346	30372	8581	0.499	0.039	0.989	0.072	0.011	0.961	0.514	0.195	0.139	0.775	0.666
Adaboost	27	28	30690	8899	0.491	0.003	0.999	0.006	0.001	0.997	0.501	0.055	0.039	0.775	0.64
Adaboost [Noise + Imputation]	0	1	30717	8926	0	0	1	NaN	0	1	0.5	0	0	0.775	0.64
Naives Bayes	4311	9205	21513	4615	0.319	0.483	0.7	0.384	0.3	0.517	0.592	0.582	0.392	0.651	0.633
C4.5 [Noise + Imputation]	1149	1837	28881	7777	0.385	0.129	0.94	0.193	0.06	0.871	0.534	0.348	0.223	0.757	0.626
C4.5	978	1532	29186	7948	0.39	0.11	0.95	0.171	0.05	0.89	0.53	0.323	0.207	0.761	0.6
KNN [Noise + Imputation]	1919	3715	27003	7007	0.341	0.215	0.879	0.264	0.121	0.785	0.547	0.435	0.271	0.73	0.587
KNN	1704	3369	23400	6059	0.336	0.22	0.874	0.266	0.126	0.78	0.547	0.438	0.272	0.727	0.563
ONER	0	0	30718	8926	?	0	1	?	0	1	0.5	0	?	0.775	0.5

El resultado obtenido era el esperado, todos los algoritmos sin excepción arrojan mejores medidas tanto en AUC, como en $F1$, aunque ninguno consigue superar a la versión con ruido de otro algoritmo, es decir, ha habido mejora *intra-algoritmo* pero no *inter-algoritmo*.

5.2. Submuestreo

En un acercamiento preliminar a esta práctica intenté aplicar *SMOTE* como estrategia de sobremuestreo para equilibrar la distribución de clases, pero el algoritmo era escandalosamente lento y arrojó además peores resultados que con submuestreo, es por eso éste método es el único que comparo de los incluidos en KNIME.



He configurado el nodo para tomar equilibrar las dos clases de forma exacta, conservando todos los ejemplos positivos y muestreando con una distribución uniforme los ejemplos negativos.

En la sección de contenido adicional analizaremos otros dos algoritmos que implementé para selección de instancias.

5.2.1. Resultados en los algoritmos

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
Gradient Boosted [Noise + Imputation]	574	676	30042	8352	0.459	0.064	0.978	0.113	0.022	0.936	0.521	0.251	0.172	0.772	0.685
Gradient Boosted	348	395	30323	8578	0.468	0.039	0.987	0.072	0.013	0.961	0.513	0.196	0.135	0.774	0.684
Gradient Boosted [Noise + Imputation + Downsample]	5545	11193	19525	3381	0.331	0.621	0.636	0.432	0.364	0.379	0.628	0.628	0.454	0.632	0.678
Random Forest [Noise + Imputation]	337	399	30319	8589	0.458	0.038	0.987	0.07	0.013	0.962	0.512	0.193	0.131	0.773	0.674
Random Forest [Noise + Imputation + Downsample]	5572	11395	19323	3354	0.328	0.624	0.629	0.43	0.371	0.376	0.627	0.627	0.453	0.628	0.673
Random Forest	416	429	30289	8510	0.492	0.047	0.986	0.085	0.014	0.953	0.516	0.214	0.151	0.775	0.673
MLP [Noise + Imputation]	385	409	30309	8541	0.485	0.043	0.987	0.079	0.013	0.957	0.515	0.206	0.145	0.774	0.667
MLP [Noise + Imputation + Downsample]	5536	11206	19512	3390	0.331	0.62	0.635	0.431	0.365	0.38	0.628	0.628	0.453	0.632	0.667
MLP [Missing: Linear Interpol/MFV + Mean]	345	346	30372	8581	0.499	0.039	0.989	0.072	0.011	0.961	0.514	0.195	0.139	0.775	0.666
Adaboost [Noise + Imputation + Downsample]	5089	11094	19624	3837	0.314	0.57	0.639	0.405	0.361	0.43	0.604	0.604	0.423	0.623	0.643
Adaboost	27	28	30690	8899	0.491	0.003	0.999	0.006	0.001	0.997	0.501	0.055	0.039	0.775	0.64
Adaboost [Noise + Imputation]	0	1	30717	8926	0	0	1	NaN	0	1	0.5	0	0	0.775	0.64
Naives Bayes	4311	9205	21513	4615	0.319	0.483	0.7	0.384	0.3	0.517	0.592	0.582	0.392	0.651	0.633
C4.5 [Noise + Imputation]	1149	1837	28881	7777	0.385	0.129	0.94	0.193	0.06	0.871	0.534	0.348	0.223	0.757	0.626
C4.5 [Noise + Imputation + Downsample]	5302	12564	18154	3624	0.297	0.594	0.591	0.396	0.409	0.406	0.592	0.592	0.42	0.592	0.601
C4.5	978	1532	29186	7948	0.39	0.11	0.95	0.171	0.05	0.89	0.53	0.323	0.207	0.761	0.6
KNN [Noise + Imputation + Downsample]	4979	12783	17935	3947	0.28	0.558	0.584	0.373	0.416	0.442	0.571	0.571	0.395	0.578	0.593
KNN [Noise + Imputation]	1919	3715	27003	7007	0.341	0.215	0.879	0.264	0.121	0.785	0.547	0.435	0.271	0.73	0.587
KNN	1704	3369	23400	6059	0.336	0.22	0.874	0.266	0.126	0.78	0.547	0.438	0.272	0.727	0.563
ONER	0	0	30718	8926	?	0	1	?	0	1	0.5	0	?	0.775	0.5

En este caso, únicamente son Adaboost y KNN los que obtienen mejores resultados en *AUC* pero sin embargo, el valor para la medida $F1$ mejora en todos, esto no es sorpresa, pues mientras que AUC es una medida sobre la totalidad de la muestra, $F1$ es una medida parcial que evalua cómo de bien clasificamos los casos positivos, al estimular la predicción de noticias *populares* incentivamos también la incorrecta clasificación de noticias que en realidad eran *no populares*, sin embargo a la luz de los datos parece que la mejora en $F1$ es significativamente mayor que el peor rendimiento para *AUC*.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
Gradient Boosted [Noise + Imputation]	574	676	30042	8352	0.459	0.064	0.978	0.113	0.022	0.936	0.521	0.251	0.172	0.772	0.685
Gradient Boosted	348	395	30323	8578	0.468	0.039	0.987	0.072	0.013	0.961	0.513	0.196	0.135	0.774	0.684
Gradient Boosted [Noise + Imputation + Downsample]	5545	11193	19525	3381	0.331	0.621	0.636	0.432	0.364	0.379	0.628	0.628	0.454	0.632	0.678
Random Forest [Noise + Imputation]	337	399	30319	8589	0.458	0.038	0.987	0.07	0.013	0.962	0.512	0.193	0.131	0.773	0.674
Random Forest [Noise + Imputation + Downsample]	5572	11395	19323	3354	0.328	0.624	0.629	0.43	0.371	0.376	0.627	0.627	0.453	0.628	0.673
Random Forest	416	429	30289	8510	0.492	0.047	0.986	0.085	0.014	0.953	0.516	0.214	0.151	0.775	0.673
MLP [Noise + Imputation]	385	409	30309	8541	0.485	0.043	0.987	0.079	0.013	0.957	0.515	0.206	0.145	0.774	0.667
MLP [Noise + Imputation + Downsample]	5536	11206	19512	3390	0.331	0.62	0.635	0.431	0.365	0.38	0.628	0.628	0.453	0.632	0.667
MLP [Missing: Linear Interpol/MFV + Mean]	345	346	30372	8581	0.499	0.039	0.989	0.072	0.011	0.961	0.514	0.195	0.139	0.775	0.666
Adaboost [Noise + Imputation + Downsample]	5089	11094	19624	3837	0.314	0.57	0.639	0.405	0.361	0.43	0.604	0.604	0.423	0.623	0.643
Adaboost	27	28	30690	8899	0.491	0.003	0.999	0.006	0.001	0.997	0.501	0.055	0.039	0.775	0.64
Adaboost [Noise + Imputation]	0	1	30717	8926	0	0	1	NaN	0	1	0.5	0	0	0.775	0.64
Naives Bayes (NR + IMP + DS)	6895	17678	13040	2031	0.281	0.772	0.425	0.412	0.575	0.228	0.598	0.573	0.466	0.503	0.636
Naives Bayes	4311	9205	21513	4615	0.319	0.483	0.7	0.384	0.3	0.517	0.592	0.582	0.392	0.651	0.633
C4.5 [Noise + Imputation]	1149	1837	28881	7777	0.385	0.129	0.94	0.193	0.06	0.871	0.534	0.348	0.223	0.757	0.626
C4.5 [Noise + Imputation + Downsample]	5302	12564	18154	3624	0.297	0.594	0.591	0.396	0.409	0.406	0.592	0.592	0.42	0.592	0.601
C4.5	978	1532	29186	7948	0.39	0.11	0.95	0.171	0.05	0.89	0.53	0.323	0.207	0.761	0.6
KNN [Noise + Imputation + Downsample]	4979	12783	17935	3947	0.28	0.558	0.584	0.373	0.416	0.442	0.571	0.571	0.395	0.578	0.593
KNN [Noise + Imputation]	1919	3715	27003	7007	0.341	0.215	0.879	0.264	0.121	0.785	0.547	0.435	0.271	0.73	0.587
KNN	1704	3369	23400	6059	0.336	0.22	0.874	0.266	0.126	0.78	0.547	0.438	0.272	0.727	0.563
ONER	0	0	30718	8926	?	0	1	?	0	1	0.5	0	?	0.775	0.5

6. Interpretación de resultados

row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEASURE	ACC	AUC
Gradient Boosted [Noise + Imputation]	574	676	30042	8352	0.459	0.064	0.978	0.113	0.022	0.936	0.521	0.251	0.172	0.772	0.685
Gradient Boosted	348	395	30323	8578	0.468	0.039	0.987	0.072	0.013	0.961	0.513	0.196	0.135	0.774	0.684
Gradient Boosted [Noise + Imputation + Downsampl	5545	11193	19525	3381	0.331	0.621	0.636	0.432	0.364	0.379	0.628	0.628	0.454	0.632	0.678
Random Forest [Noise + Imputation]	337	399	30319	8589	0.458	0.038	0.987	0.070	0.013	0.962	0.512	0.193	0.131	0.773	0.674
Random Forest [Noise + Imputation + Downsample]	5572	11395	19323	3354	0.328	0.624	0.629	0.430	0.371	0.376	0.627	0.627	0.453	0.628	0.673
Random Forest	416	429	30289	8510	0.492	0.047	0.986	0.085	0.014	0.953	0.516	0.214	0.151	0.775	0.673
MLP [Noise + Imputation]	385	409	30309	8541	0.485	0.043	0.987	0.079	0.013	0.957	0.515	0.206	0.145	0.774	0.667
MLP [Noise + Imputation + Downsample]	5536	11206	19512	3390	0.331	0.620	0.635	0.431	0.365	0.380	0.628	0.628	0.453	0.632	0.667
MLP (Missing: Linear Interpol/MFV + Mean)	345	346	30372	8581	0.499	0.039	0.989	0.072	0.011	0.961	0.514	0.195	0.139	0.775	0.666
Adaboost [Noise + Imputation + Downsample]	5089	11094	19624	3837	0.314	0.570	0.639	0.405	0.361	0.430	0.604	0.604	0.423	0.623	0.643
Adaboost	27	28	30690	8899	0.491	0.003	0.999	0.006	0.001	0.997	0.501	0.055	0.039	0.775	0.640
Adaboost [Noise + Imputation]	0	1	30717	8926	0.000	0.000	1.000	NaN	0.000	1.000	0.500	0.000	0.000	0.775	0.640
Naives Bayes (NR + IMP + DS)	6895	17678	13040	2031	0.281	0.772	0.425	0.412	0.575	0.228	0.598	0.573	0.466	0.503	0.636
Naives Bayes	4311	9205	21513	4615	0.319	0.483	0.700	0.384	0.300	0.517	0.592	0.582	0.392	0.651	0.633
C4.5 [Noise + Imputation]	1149	1837	28881	7777	0.385	0.129	0.940	0.193	0.060	0.871	0.534	0.348	0.223	0.757	0.626
C4.5 [Noise + Imputation + Downsample]	5302	12564	18154	3624	0.297	0.594	0.591	0.396	0.409	0.406	0.592	0.592	0.420	0.592	0.601
C4.5	978	1532	29186	7948	0.390	0.110	0.950	0.171	0.050	0.890	0.530	0.323	0.207	0.761	0.600
KNN [Noise + Imputation + Downsample]	4979	12783	17935	3947	0.280	0.558	0.584	0.373	0.416	0.442	0.571	0.571	0.395	0.578	0.593
KNN [Noise + Imputation]	1919	3715	27003	7007	0.341	0.215	0.879	0.264	0.121	0.785	0.547	0.435	0.271	0.730	0.587
KNN	1704	3369	23400	6059	0.336	0.220	0.874	0.266	0.126	0.780	0.547	0.438	0.272	0.727	0.563
ONER	0	0	30718	8926		0.000	1.000		0.000	1.000	0.500	0.000		0.775	0.500

Lo primero que observamos es que tenemos tres candidatos para el primer puesto de calidad similar, **Gradient Boosted [N+I+D]**, **Random Forest [N+I+D]** y **MLP [N+I+D]**, los tres ofrecen un buen rendimiento tanto para la medida del área como para *F1-score*, las dos medidas que hemos tomado como

referencia como ya comentamos por las características particulares de nuestro problema, asimismo ofrecen sus resultados están entre los mejores para *Gmean* y *Gmeasure*, indicadores también de la bondad del modelo como analizamos en su apartado.

El hecho de que los tres arrojen resultados similares —perteneciendo a familiar dispares— parece indicar que estamos agotando la capacidad predictiva para estos datos para la tecnologías que estamos utilizando, y que las mejoras, previsiblemente poco sustanciales se derivarán o bien de mejoras triviales, como aumentar el número de iteraciones en MLP y Gradient Boosted, o aumentar el número de árboles en Random Forest y Gradient Boosted; o por el contrario, de la aplicación de algoritmos sobre los datos como selección de instancias, posteriormente, en el siguiente epígrafe veremos algunas de estas estrategias.

Ninguno de los algoritmos con mejor resultado tienen una interpretación fácil o inmediata, podemos sin embargo utilizar analizar cómo correlaciona la confianza para la clase positiva con cada una de las variables de forma independiente a fin de hacernos una idea aproximada de qué está aprendiendo cada unos de los algoritmos.

	GB	RF	MLP
LDA_03	0.43	0.39	0.42
kw_min_avg	0.29	0.29	0.30
global_subjectivity	0.23	0.20	0.16
data_channel_is_socmed	0.22	0.19	0.21
num_keywords	0.19	0.16	0.25
kw_avg_max	0.17	0.17	0.14
abs_title_sentiment_polarity	0.17	0.16	0.17
sunday	0.17	0.14	0.21
saturday	0.15	0.14	0.17
title_subjectivity	0.15	0.14	0.15
global_rate_positive_words	0.14	0.14	0.14
data_channel_is_lifestyle	0.13	0.12	0.11
title_sentiment_polarity	0.12	0.11	0.10
avg_positive_polarity	0.11	0.10	0.09
data_channel_is_tech	0.10	0.11	0.12
max_positive_polarity	0.08	0.07	0.07
LDA_04	0.07	0.10	0.09
LDA_00	0.04	0.02	0.06
kw_min_min	0.03	0.03	0.06
rate_positive_words	0.01	0.01	0.02
abs_title_subjectivity	0.00	-0.01	0.01
friday	-0.01	-0.01	0.01
monday	-0.04	-0.03	0.00
thursday	-0.04	-0.05	-0.08
min_negative_polarity	-0.05	-0.04	-0.04
tuesday	-0.05	-0.05	-0.07
wednesday	-0.06	-0.05	-0.08
n_tokens_title	-0.08	-0.08	-0.09
data_channel_is_bus	-0.10	-0.11	-0.13
LDA_01	-0.11	-0.09	-0.08
avg_negative_polarity	-0.11	-0.10	-0.10
rate_negative_words	-0.14	-0.15	-0.17
data_channel_is_entertainment	-0.15	-0.11	-0.16
average_token_length	-0.17	-0.16	-0.17
data_channel_is_world	-0.47	-0.45	-0.48

Vemos, a la luz de esta tabla como los tres algoritmos parecen coincidir de forma general —salvo contadas excepciones con qué variables son más determinantes

para clasificar una instancia en una clase u otra. Hay algunas observaciones más evidentes y otras más sorprendentes. En la linea de la primera categoría —evidentes—, parece que los domingos y los sábados son un buen día para estimular la popularidad, cabe pensar que en los días de descanso —fin de semana— la gente tiene más tiempo para compartir noticias, también que los artículos de la categoría *Social Media* se estiman como más populares, de la misma forma que *World* y *Entertainment* tienen menos feedback por parte del público; o que la subjetividad del artículo despierta más interés en el público a la hora de compartir. Sin embargo, otras menos evidentes es que la medida *kw_min_avg* —el artículo con la etiqueta media menos compartido— se estima que correlaciona positivamente con la popularidad, esto tiene sentido, si de todas las etiquetas que tiene una noticia, nos quedamos con la compartida un número de veces medio, y todos los artículos con esa etiqueta han sido compartidos un número alto de veces, cabe pensar que esta noticia suscitará el mismo interés; una variable al respecto de la cual no se ponen de acuerdo los algoritmos en importancia es el número de palabras claves, todos estiman que a mayor número mayor probabilidad de popularidad, pero sin embargo, MLP lo estima de una forma más significativa que el resto; finalmente, parece que la variable más potente para estimar la popularidad de una noticia es la cercanía máxima a la tercera categoría extraida mediante *LDA*, pero no conocemos su significado concreto, mientras que las otras tres que pasaron el filtro de correlación y varianza no parecen demasiado significativas. Si bien este análisis es interesante no podemos ver la correlación cruzada entre dos variables y la predicción de salida, lo cual parece determinante a la hora de clasificar las noticias, pues como vimos en el primer apartado con el gráfico de coordenadas paralelas, no parece un problema linealmente separable, que se pueda expresar como combinación lineal de las características de entrada.

Es interesante ver, como comentábamos al principio de este trabajo, qué sacrificio cuantitativo hemos realizado al aplicar los filtros iniciales de correlación y baja varianza, vamos a analizar los tres algoritmos con mejor resultado sobre el conjunto completo de las características.

	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEASURE	ACC	AUC	GANANCIA
[Gradient Boosted [ALL + Noise + Imputation + Downsample	6008	10996	19722	2918	0,353	0,673	0,642	0,463	0,358	0,327	0,658	0,657	0,488	0,649	0,711	1,049
Random Forest [ALL + Noise + Imputation + Downsample]	5823	10975	19743	3103	0,347	0,652	0,643	0,453	0,357	0,348	0,648	0,648	0,476	0,645	0,700	1,040
MLP [ALL + Noise + Imputation + Downsample]	5890	11339	19379	3036	0,342	0,660	0,631	0,450	0,369	0,340	0,645	0,645	0,475	0,637	0,695	1,042
Gradient Boosted [Noise + Imputation + Downsample]	5545	11193	19525	3381	0,331	0,621	0,636	0,432	0,364	0,379	0,628	0,628	0,454	0,632	0,678	
Random Forest [Noise + Imputation + Downsample]	5572	11395	19323	3354	0,328	0,624	0,629	0,430	0,371	0,376	0,627	0,627	0,453	0,628	0,673	
MLP [Noise + Imputation + Downsample]	5536	11206	19512	3390	0,331	0,620	0,635	0,431	0,365	0,380	0,628	0,628	0,453	0,632	0,667	

Más adelante podemos verlos junto al resto en la lista completa de resultados, pero ahora observamos, tal y como esperaba, que los resultados son mejores, pero lo interesante es ver que la ganancia en los tres casos, comparando el area bajo la curva es similar, lo que nos da una idea aproximada del sacrificio que hemos hecho durante el experimento a fin de agilizar el proceso. Cabe esperar que la ganancia sobre el resto de algoritmos sea similiar a la luz de estos resultados, especialmente al ver cómo las tres familias de algoritmos —arboles con boosting, arboles con bagging y redes neuronales— experimentan una ganancia similar al ser entrenados con el conjunto completo de datos.

Una prueba que no he hecho en esta práctica es aplicar *Principal Component Analysis* a fin de reducir la dimensionalidad del problema, idealmente conservando

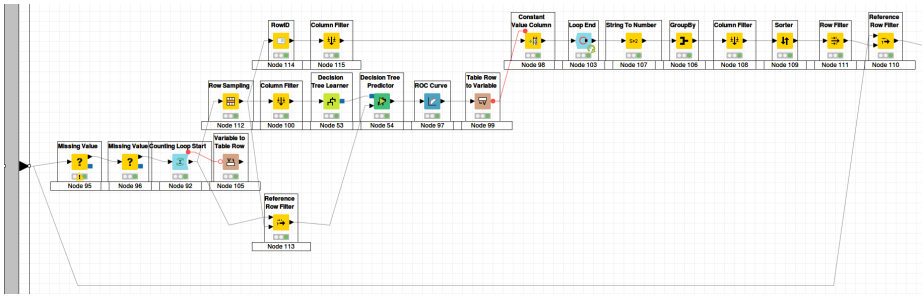
toda nuestra capacidad predictiva. El motivo de esto es conservar la interpretabilidad de los resultados a fin de responder a éste el último apartado de la práctica. Otro preprocesado que me habría gustado tener oportunidad de probar era la discretización de las variables, pero las dos opciones que vi que me ofrecía KNIME —discretización por igual frecuencia o igual amplitud— en experimentaciones preliminares vi que producían sistemáticamente peores resultados, tanto antes como después del equilibrado de datos, es posible que estas estrategias tan ingenuas no sean capaz de representar la naturaleza de los datos con suficiente capacidad expresiva.

7. Contenido adicional

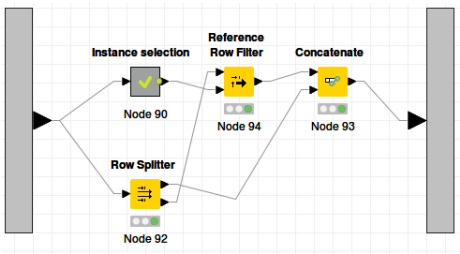
7.1. Condensed Nearest Neighbor (CNN)

Implementé el algoritmo CNN a través de la extensión de Python pero el problema resultó ser que el *parser* de KNIME que comunica los dos lenguajes se congela para aproximadamente más de 2000 instancias, así que para las 30000 que recibía 5CV en cada iteración el algoritmo no era capaz de arrojar ningún resultado, y evaluarlo sobre una muestra aleatoria de menos de 2000 instancias parecía un intento absurdo si lo que intentamos es medir la calidad del método. Ya tendremos ocasión de experimentar con este algoritmo en la siguiente práctica.

7.2. Smart instance selection



Ante el fracaso del algoritmo anterior implementé en KNIME mi propio algoritmo de selección de instancias —muy ingenuo— intentando conseguir así clasificar mejor el problema. El procedimiento es sencillo, Realizo en 10 iteraciones un sampleo aleatorio, del 50 % de las instancias, entreno C4.5 con éstas y evaluo sobre la totalidad de la muestra, finalmente calculo para cada instancia la media del AUC para todos los modelos en los que ha participado como entrenamiento, las ordeno por este valor y selecciono aproximadamente la mitad superior —15000 instancias—. Finalmente tomo la totalidad de las instancias con clase positiva y sólo las mejores de la clase negativa.

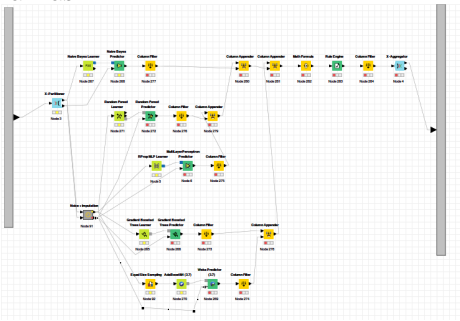


Con este algoritmo conseguí seleccionar únicamente la mitad de las características, y conservar toda la capacidad predictiva, incluso mejorarla ligeramente en el caso del Random Forest, pero a mi entender, el incremente muy significativo de tiempo no va parejo al incremento en el rendimiento del sistema.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR.	Are...
Gradient Boosted [Noise + Imputation]	574	676	30042	8352	0.459	0.064	0.978	0.113	0.022	0.936	0.521	0.251	0.172	0.772	0.685
Gradient Boosted	348	395	30323	8578	0.468	0.039	0.987	0.072	0.013	0.961	0.513	0.196	0.135	0.774	0.684
Gradient Boosted [Noise + Imputation + Smart Downsample]	2923	4422	26296	6003	0.398	0.327	0.856	0.359	0.144	0.673	0.592	0.529	0.361	0.737	0.681
Gradient Boosted [Noise + Imputation + Downsample]	5545	11193	19525	3381	0.331	0.621	0.636	0.432	0.364	0.379	0.628	0.628	0.454	0.632	0.678
Random Forest [Noise + Imputation + Smart Downsample]	2809	4434	26284	6117	0.388	0.315	0.856	0.347	0.144	0.685	0.585	0.519	0.349	0.734	0.674
Random Forest [Noise + Imputation]	337	399	30319	8589	0.458	0.038	0.987	0.07	0.013	0.962	0.512	0.193	0.131	0.773	0.674
Random Forest [Noise + Imputation + Downsample]	5572	11395	19323	3354	0.328	0.624	0.629	0.43	0.371	0.376	0.627	0.627	0.453	0.628	0.673
Random Forest	416	429	30289	8510	0.492	0.047	0.986	0.085	0.014	0.953	0.516	0.214	0.151	0.775	0.673
MLP [Noise + Imputation]	385	409	30309	8541	0.485	0.043	0.987	0.079	0.013	0.957	0.515	0.206	0.145	0.774	0.667
MLP [Noise + Imputation + Smart Downsample]	3286	5201	25517	5640	0.387	0.368	0.831	0.377	0.169	0.632	0.599	0.553	0.378	0.727	0.667
MLP [Noise + Imputation + Downsample]	5536	11206	19512	3390	0.331	0.62	0.635	0.431	0.365	0.38	0.628	0.628	0.453	0.632	0.667
MLP (Missing: Linear Interpol/MFV + Mean)	345	346	30372	8581	0.499	0.039	0.989	0.072	0.011	0.961	0.514	0.195	0.139	0.775	0.666
Adaboost [Noise + Imputation + Downsample]	5089	11094	19624	3837	0.314	0.57	0.639	0.405	0.361	0.43	0.604	0.604	0.423	0.623	0.643
Adaboost	27	28	30690	8899	0.491	0.003	0.999	0.006	0.001	0.997	0.501	0.055	0.039	0.775	0.64
Adaboost [Noise + Imputation]	0	1	30717	8926	0	0	1	NaN	0	1	0.5	0	0	0.775	0.64
Naives Bayes	4311	9205	21513	4615	0.319	0.483	0.7	0.384	0.3	0.517	0.592	0.582	0.392	0.651	0.633
Adaboost [Noise + Imputation + Smart Downsample]	1971	3262	27456	6955	0.377	0.221	0.894	0.278	0.106	0.779	0.557	0.444	0.288	0.742	0.633
C4.5 [Noise + Imputation]	1149	1837	28881	7777	0.385	0.129	0.94	0.193	0.06	0.871	0.534	0.348	0.223	0.757	0.626
C4.5 [Noise + Imputation + Smart Downsample]	3425	7055	23663	5501	0.327	0.384	0.77	0.353	0.23	0.616	0.577	0.544	0.354	0.683	0.611
C4.5 [Noise + Imputation + Downsample]	5302	12564	18154	3624	0.297	0.594	0.591	0.396	0.409	0.406	0.592	0.592	0.42	0.592	0.601
kNN [Smart Downsample]	3693	8237	22481	5233	0.31	0.414	0.732	0.354	0.268	0.586	0.573	0.55	0.358	0.66	0.601
C4.5	978	1532	29186	7948	0.39	0.11	0.95	0.171	0.05	0.89	0.53	0.323	0.207	0.761	0.6
kNN [Noise + Imputation + Downsample]	4979	12783	17935	3947	0.28	0.558	0.584	0.373	0.416	0.442	0.571	0.571	0.395	0.578	0.593
kNN [Noise + Imputation]	1919	3715	27003	7007	0.341	0.215	0.879	0.264	0.121	0.785	0.547	0.435	0.271	0.73	0.587
kNN	1704	3369	23400	6059	0.336	0.22	0.874	0.266	0.126	0.78	0.547	0.438	0.272	0.727	0.563
ONER	0	0	30718	8926	?	0	1	?	0	1	0.5	0	?	0.775	0.5

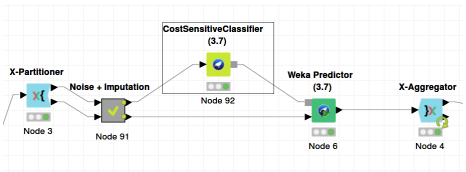
7.3. Multiclasificador

La siguiente idea fue realizar un multiclasificador, utilizando los cinco clasificadores con sus configuraciones más exitosas y que votaran la clase ganadora calculando la media de las confianzas.



Sin embargo, los resultados, como veremos en la tabla final no conseguían ser mejores que el mejor de los 5. El valor para el área es competitivo pero la votación lo único que consigue es premiar a la clase dominante, disminuyendo el valor para $F1$, pues si uno ha sido capaz de estimar correctamente que pertenece a la clase positiva el resto de votos tienden a ahogarlo.

7.4. Pesos no uniformes en la matriz de confusión



La siguiente idea fue utilizar pesos no uniformes en la matriz de confusión en vez de realizar *downsample* para compensar el tamaño desigual de las clases, dándole proporcionalmente más peso a los errores para la clase negativa —instancias realmente negativas clasificadas como positivas— que a las contrarias, así remediamos la ventaja de los algoritmos al beneficiar a la clase dominante. Sin embargo, el rendimiento para mi sorpresa fue significativamente peor que en los casos con *downsample* como estrategia de balanceo. El algoritmo de aprendizaje que subyacía era MLP con los mismo parámetros que en su versión regular.

Row ID	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEA...	ACCUR...	Are...
Gradient Boosted [Noise + Imputation + Downsample]	5545	11193	19525	3381	0.331	0.621	0.636	0.432	0.364	0.379	0.628	0.628	0.454	0.632	0.678
Random Forest [Noise + Imputation + Downsample]	5572	11395	19323	3354	0.328	0.624	0.629	0.43	0.371	0.376	0.627	0.627	0.453	0.628	0.673
MLP [Noise + Imputation + Downsample]	5536	11206	19512	3390	0.331	0.62	0.635	0.431	0.365	0.38	0.628	0.628	0.453	0.632	0.667
Adaboost [Noise + Imputation + Downsample]	5089	11094	19624	3837	0.314	0.57	0.639	0.405	0.361	0.43	0.604	0.604	0.423	0.623	0.643
MLP WEIGTH	2148	3968	26750	6778	0.351	0.241	0.871	0.286	0.129	0.759	0.556	0.458	0.291	0.729	0.643
C4.5 [Noise + Imputation + Downsample]	5302	12564	18154	3624	0.297	0.594	0.591	0.396	0.409	0.406	0.592	0.592	0.42	0.592	0.601
KNN [Noise + Imputation + Downsample]	4979	12783	17935	3947	0.28	0.558	0.584	0.373	0.416	0.442	0.571	0.571	0.395	0.578	0.593

8. Tabla con todos los resultados

	TP	FP	TN	FN	PPV	TPR	TNR	F1	FPR	FNR	AUC	GMEAN	GMEASURE	ACC	AUC
[Gradient Boosted [ALL + Noise + Imputation + Downsample	6008	10996	19722	2918	0,353	0,673	0,642	0,463	0,358	0,327	0,658	0,657	0,488	0,649	0,711
Random Forest [ALL + Noise + Imputation + Downsample]	5823	10975	19743	3103	0,347	0,652	0,643	0,453	0,357	0,348	0,648	0,648	0,476	0,645	0,700
MLP [ALL + Noise + Imputation + Downsample]	5890	11339	19379	3036	0,342	0,660	0,631	0,450	0,369	0,340	0,645	0,645	0,475	0,637	0,695
Gradient Boosted [Noise + Imputation]	574	676	30042	8352	0,459	0,064	0,978	0,113	0,022	0,936	0,521	0,251	0,172	0,772	0,685
Gradient Boosted	348	395	30323	8578	0,468	0,039	0,987	0,072	0,013	0,961	0,513	0,196	0,135	0,774	0,684
[Gradient Boosted [Noise + Imputation + Smart Downsample	2923	4422	26296	6003	0,398	0,327	0,856	0,359	0,144	0,673	0,592	0,529	0,361	0,737	0,681
Gradient Boosted [Noise + Imputation + Downsample]	5545	11193	19525	3381	0,331	0,621	0,636	0,432	0,364	0,379	0,628	0,628	0,454	0,632	0,678
Random Forest [Noise + Imputation + Smart Downsample]	2809	4434	26284	6117	0,388	0,315	0,856	0,347	0,144	0,685	0,585	0,519	0,349	0,734	0,674
Random Forest [Noise + Imputation]	337	399	30319	8589	0,458	0,038	0,987	0,070	0,013	0,962	0,512	0,193	0,131	0,773	0,674
Random Forest [Noise + Imputation + Downsample]	5572	11395	19323	3354	0,328	0,624	0,629	0,430	0,371	0,376	0,627	0,627	0,453	0,628	0,673
Random Forest	416	429	30289	8510	0,492	0,047	0,986	0,085	0,014	0,953	0,516	0,214	0,151	0,775	0,673
Multiclas	1901	2567	28151	7025	0,425	0,213	0,916	0,284	0,084	0,787	0,565	0,442	0,301	0,758	0,671
MLP [Noise + Imputation]	385	409	30309	8541	0,485	0,043	0,987	0,079	0,013	0,957	0,515	0,206	0,145	0,774	0,667
MLP [Noise + Imputation + Smart Downsample]	3286	5201	25517	5640	0,387	0,368	0,831	0,377	0,169	0,632	0,599	0,553	0,378	0,727	0,667
MLP [Noise + Imputation + Downsample]	5536	11206	19512	3390	0,331	0,620	0,635	0,431	0,365	0,380	0,628	0,628	0,453	0,632	0,667
MLP (Missing: Linear Interpol/MFV + Mean)	345	346	30372	8581	0,499	0,039	0,989	0,072	0,011	0,961	0,514	0,195	0,139	0,775	0,666
Adaboost [Noise + Imputation + Downsample]	5089	11094	19624	3837	0,314	0,570	0,639	0,405	0,361	0,430	0,604	0,604	0,423	0,623	0,643
MLP WEIGTH	2148	3968	26750	6778	0,351	0,241	0,871	0,286	0,129	0,759	0,556	0,458	0,291	0,729	0,643
Adaboost	27	28	30690	8899	0,491	0,003	0,999	0,006	0,001	0,997	0,501	0,055	0,039	0,775	0,640
Adaboost [Noise + Imputation]	0	1	30717	8926	0,000	0,000	1,000	NaN	0,000	1,000	0,500	0,000	0,000	0,775	0,640
Naives Bayes (NR + IMP + DS)	6895	17678	13040	2031	0,281	0,772	0,425	0,412	0,575	0,228	0,598	0,573	0,466	0,503	0,636
Naives Bayes	4311	9205	21513	4615	0,319	0,483	0,700	0,384	0,300	0,517	0,592	0,582	0,392	0,651	0,633
Adaboost [Noise + Imputation + Smart Downsample]	1971	3262	27456	6955	0,377	0,221	0,894	0,278	0,106	0,779	0,557	0,444	0,288	0,742	0,633
C4.5 [Noise + Imputation]	1149	1837	28881	7777	0,385	0,129	0,940	0,193	0,060	0,871	0,534	0,348	0,223	0,757	0,626
C4.5 [Noise + Imputation + Smart Downsample]	3425	7055	23663	5501	0,327	0,384	0,770	0,353	0,230	0,616	0,577	0,544	0,354	0,683	0,611
C4.5 [Noise + Imputation + Downsample]	5302	12564	18154	3624	0,297	0,594	0,591	0,396	0,409	0,406	0,592	0,592	0,420	0,592	0,601
KNN [Smart Downsample]	3693	8237	22481	5233	0,310	0,414	0,732	0,354	0,268	0,586	0,573	0,550	0,358	0,660	0,601
C4.5	978	1532	29186	7948	0,390	0,110	0,950	0,171	0,050	0,890	0,530	0,323	0,207	0,761	0,600
KNN [Noise + Imputation + Downsample]	4979	12783	17935	3947	0,280	0,558	0,584	0,373	0,416	0,442	0,571	0,571	0,395	0,578	0,593
KNN [Noise + Imputation]	1919	3715	27003	7007	0,341	0,215	0,879	0,264	0,121	0,785	0,547	0,435	0,271	0,730	0,587
KNN	1704	3369	23400	6059	0,336	0,220	0,874	0,266	0,126	0,780	0,547	0,438	0,272	0,727	0,563
ONER	0	0	30718	8926		0,000	1,000		0,000	1,000	0,500	0,000		0,775	0,500

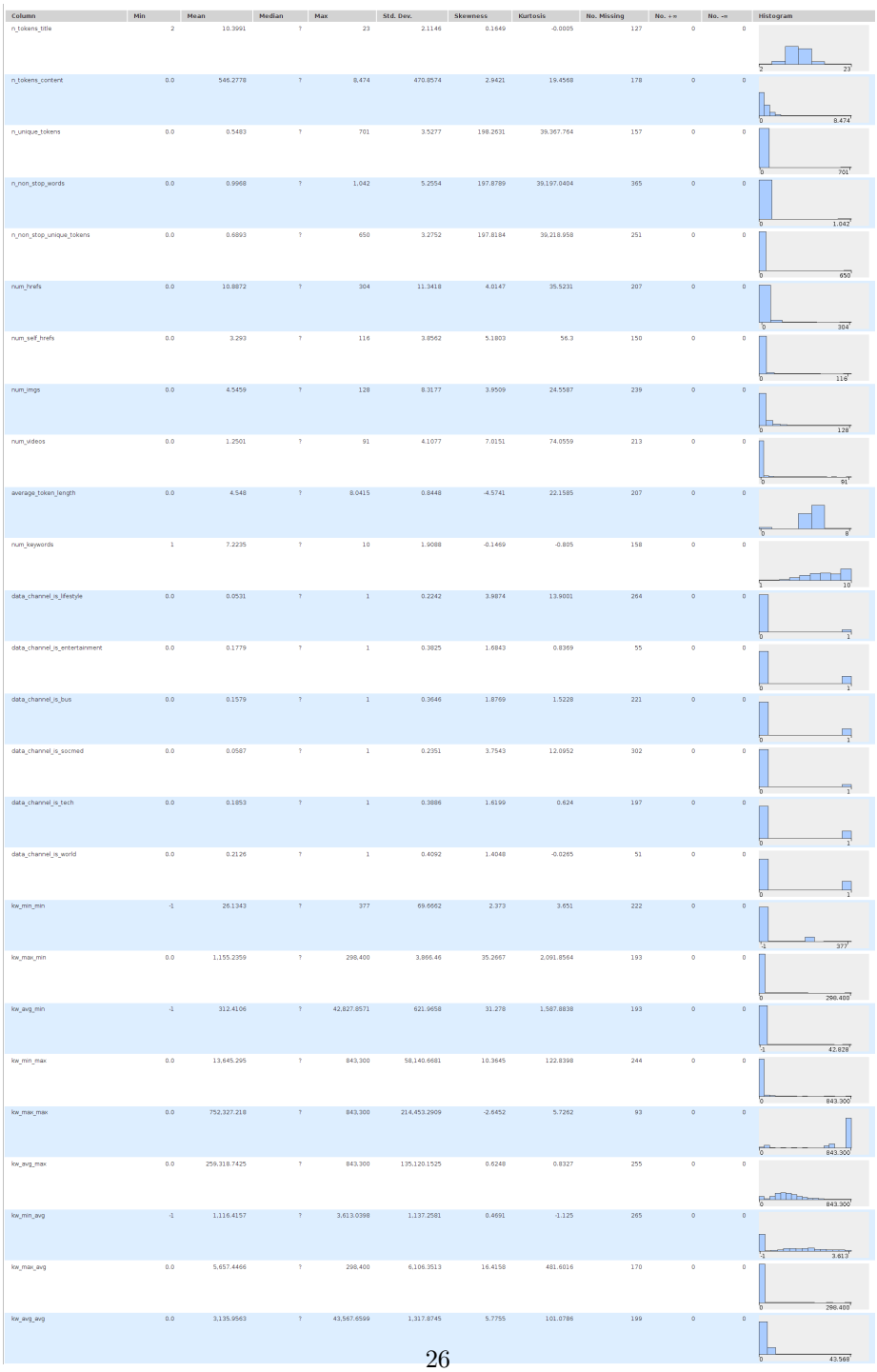
9. Bibliografía

1. FERNANDES, Kelwin. (2015). A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News. 10.1007/978-3-319-23485-4_53. Accedido el 4 de Noviembre, `https://pdfs.semanticscholar.org/ad7f/3da7a5d6a1e18cc5a176f18f52687b912fea.pdf`

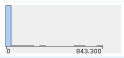
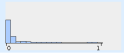
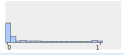
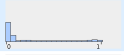
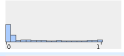
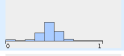
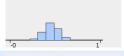
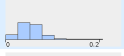
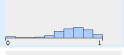
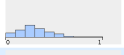
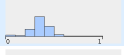
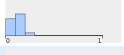
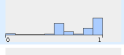
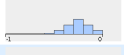
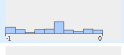
2. BLEI, David M., NG, Andrew Y., and JORDAN, Michael I. (2003). Latent dirichlet allocation. J. Mach. Learn. Res. 3 (March 2003), 993-1022. Accedido el 4 de Noviembre, `https://ai.stanford.edu/~ang/papers/nips01-lda.pdf`

10. Apéndice

10.1. Estadísticas datos



Column	Min	Mean	Median	Max	Std. Dev.	Skewness	Kurtosis	No. Missing	No. +=	No. -=	Histogram
n_tokens_title	2	10.3991	?	23	2.1146	0.1649	-0.0005	127	0	0	2 23
n_tokens_content	0.0	546.2778	?	8,474	470.8574	2.9421	19.4568	178	0	0	0 8,474
n_unique_tokens	0.0	0.5483	?	701	3.5277	198.2631	39,367.764	157	0	0	0 701
n_non_stop_words	0.0	0.9968	?	1,042	5.2554	197.8789	39,197.0404	365	0	0	0 1,042
n_non_stop_unique_tokens	0.0	0.6893	?	650	3.2752	197.8184	39,218.958	251	0	0	0 650
num_hrefs	0.0	10.8872	?	304	11.3418	4.0147	35.5231	207	0	0	0 304
num_self_hrefs	0.0	3.293	?	116	3.8562	5.1803	56.3	150	0	0	0 116
num_imgs	0.0	4.5459	?	128	8.3177	3.9509	24.5587	239	0	0	0 128
num_videos	0.0	1.2501	?	91	4.1077	7.0151	74.0559	213	0	0	0 91
average_token_length	0.0	4.548	?	8.0415	0.8448	-4.5741	22.1585	207	0	0	0 8
num_keywords	1	7.2235	?	10	1.9088	-0.1469	-0.805	158	0	0	1 10
data_channel_is_lifestyle	0.0	0.0531	?	1	0.2242	3.9874	13.9001	264	0	0	0 1
data_channel_is_entertainment	0.0	0.1779	?	1	0.3825	1.6843	0.8369	55	0	0	0 1
data_channel_is_bus	0.0	0.1579	?	1	0.3646	1.8769	1.5228	221	0	0	0 1
data_channel_is_socmed	0.0	0.0587	?	1	0.2351	3.7543	12.0952	302	0	0	0 1
data_channel_is_tech	0.0	0.1853	?	1	0.3886	1.6199	0.624	197	0	0	0 1
data_channel_is_world	0.0	0.2126	?	1	0.4092	1.4048	-0.0265	51	0	0	0 1
kw_min_min	-1	26.1343	?	377	69.6662	2.373	3.651	222	0	0	-1 377
kw_max_min	0.0	1,155.2359	?	298,400	3,866.46	35.2667	2,091.8564	193	0	0	0 298,400
kw_avg_min	-1	312.4106	?	42,827.8571	621.9658	31.278	1,587.8838	193	0	0	-1 42,828
kw_min_max	0.0	13,645.295	?	843,300	58,140.6681	10.3645	122.8398	244	0	0	0 843,300
kw_max_max	0.0	752,327.218	?	843,300	214,453.2909	-2.6452	5.7262	93	0	0	0 843,300
kw_avg_max	0.0	259,318.7425	?	843,300	135,120.1525	0.6248	0.8327	255	0	0	0 843,300
kw_min_avg	-1	1,116.4157	?	3,613.0398	1,137.2581	0.4691	-1.125	265	0	0	-1 3,613
kw_max_avg	0.0	5,657.4466	?	298,400	6,106.3513	16.4158	481.6016	170	0	0	0 298,400
kw_avg_avg	0.0	3,135.9563	?	43,567.6599	1,317.8745	5.7755	101.0786	199	0	0	0 43,568

self_reference_min_shares	0.0	4,004.0261	?	843,300	19,813.5333	26.2103	860.0387	379	0	0	
self_reference_max_shares	0.0	10,349.7115	?	843,300	41,128.4038	13.8478	223.2644	299	0	0	
self_reference_avg_sharess	0.0	6,405.8828	?	843,300	24,268.3129	17.8932	427.0673	243	0	0	
LDA_00	0.0	0.1846	?	0.927	0.263	1.5679	1.0623	292	0	0	
LDA_01	0.0	0.1414	?	0.9259	0.2199	2.0847	3.3355	198	0	0	
LDA_02	0.0	0.2165	?	0.92	0.2822	1.3105	0.2587	218	0	0	
LDA_03	0.0	0.2239	?	0.9265	0.2953	1.2376	-0.0324	290	0	0	
LDA_04	0.0	0.2338	?	0.9272	0.289	1.1742	-0.0779	120	0	0	
global_subjectivity	0.0	0.4433	?	1	0.1167	-1.3727	4.61	186	0	0	
global_sentiment_polarity	-0.3937	0.1193	?	0.7278	0.097	0.1053	1.5144	225	0	0	
global_rate_positive_words	0.0	0.0396	?	0.1555	0.0174	0.3236	1.0231	208	0	0	
global_rate_negative_words	0.0	0.0166	?	0.1849	0.0108	1.4947	7.0005	191	0	0	
rate_positive_words	0.0	0.6821	?	1	0.1902	-1.4231	3.2748	9	0	0	
rate_negative_words	0.0	0.2879	?	1	0.1561	0.4066	0.518	183	0	0	
avg_positive_polarity	0.0	0.3539	?	1	0.1045	-0.7235	3.3857	169	0	0	
min_positive_polarity	0.0	0.0954	?	1	0.0713	3.0435	17.481	280	0	0	
max_positive_polarity	0.0	0.7566	?	1	0.2479	-0.9388	0.6597	337	0	0	
avg_negative_polarity	-1	-0.2595	?	0.0	0.1277	-0.5528	2.3784	230	0	0	
min_negative_polarity	-1	-0.5219	?	0.0	0.2902	-0.0734	-0.8263	76	0	0	
max_negative_polarity	-1	-0.1075	?	0.0	0.0952	-3.4543	19.5168	197	0	0	
title_subjectivity	0.0	0.2825	?	1	0.3243	0.8154	-0.5423	221	0	0	
title_sentiment_polarity	-1	0.0714	?	1	0.2655	0.3936	3.2344	223	0	0	
abs_title_subjectivity	0.0	0.3419	?	0.5	0.1888	-0.6252	-1.2775	141	0	0	
abs_title_sentiment_polarity	0.0	0.1562	?	1	0.2264	1.704	2.6626	218	0	0	

10.2. Matriz de correlación