ANÁLISIS PREDICTIVO MEDIANTE CLASIFICACIÓN

Ignacio Garach Vélez
igarachv@correo.ugr.es IN Viernes 12:30

9 de noviembre de 2022

Índice

1. Introducción

En esta práctica vamos a llevar a cabo la labor más común dentro del contexto del aprendizaje supervisado, clasificación mediante algoritmos de aprendizaje automático en varios datasets. La idea será llevar a cabo un preprocesado para preparar los datos para aplicar los distintos algoritmos, esto es necesario ya que dependiendo de si las variables son categóricas o numéricas necesitan cierta actuación y pueden darse casos de datos ruidosos o de outliers que pueden empeorar el desempeño de los algoritmos si no se tratan correctamente. Realizaremos una comparación entre los distintos algoritmos mediante distintas técnica y métricas que tomarán mayor o menor importancia dependiendo de las características del dataset. Se intentará utilizar variedad de algoritmos para comparar las distintas técnicas estudiadas en teoría, desde árboles de decisión, a redes neuronales, máquinas de soporte vectorial hasta ensemblers de tipo bagging o boosting que se encuentran en el actual estado del arte. En primer lugar, realizaremos un análisis exploratorio de los distintos conjuntos de datos, para ello se hara uso del nodo Statistics para comprobar valores faltantes y generaremos gráficas para entender la distribución de datos y sus correlaciones.

1.1. heart_2020

Problema de predicción de fallos cardíacos a partir de factores personales o estilo de vida, los datos han sido tomados por el centro de detección y control de enfermedades de Estados Unidos. Es por ello un problema de clasificación binaria con clases positiva y negativa, la variable a predecir es la dada por la columna HeartFailure. Tenemos una mayoría de variables categóricas, 13 en total, gran parte de ellas binarias. Por otro lado las clases estan muy desbalanceadas en favor de la negativa (sin fallo cardíaco), en la documentación se recomienda utilizar undersampling u otras técnicas para mejorar el aprendizaje. También nos damos cuenta que deberemos utilizar una métrica distinta del accuracy, ya que esta sobreestimará la calidad del modelo, un clasificador que devuelva siempre la clase mayoritaria tendrá buena accuracy pero será inútil y de hecho peligroso, ya que no predecirá ningún futuro enfermo.

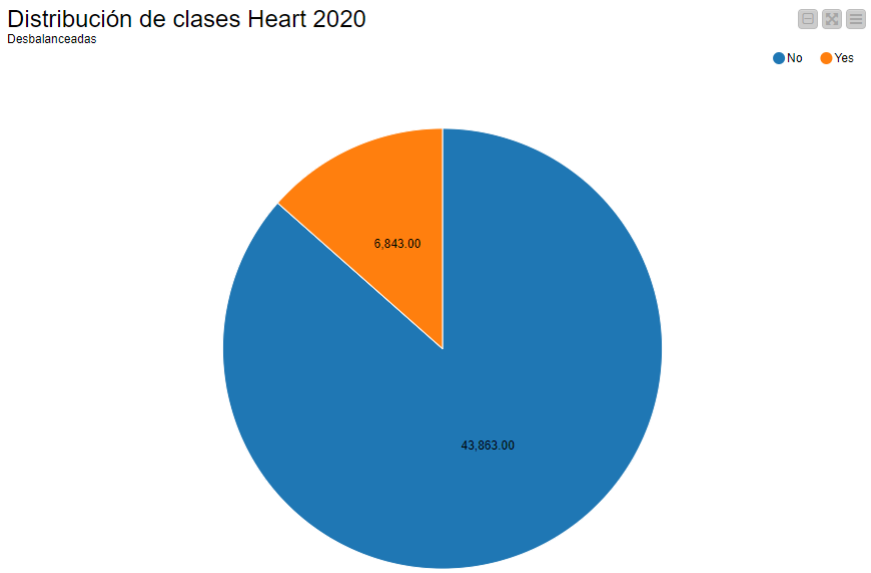


Figura 1: Distribución desbalanceada de clases

Vamos a mostrar los promedios de ciertas variables separados por la clase a la que pertenecen, desde ahí podremos hacernos una idea de la influencia de las variables, en este caso, fumadores, bebedores y antecedentes cardíacos.

Figura 2: Promedio de variables por clase

Parece claro que tanto las variables relacionadas con el tabaco y un pasado ataque cardíaco son más altas en los pacientes de la clase positiva, luego serán importantes en la predicción, la variable relacionada con el consumo de alcohol sin embargo es similar en ambos grupos y no parece que vaya a ser capital en el análisis.

Finalmente, vamos a generar una matriz de correlaciones entre las variables para visualizar en franjas de color la relación entre las mismas, si son altas podemos plantearnos realizar un preprocesado en el que eliminar algunas de ellas o reducirlas mediante técnicas como el análisis de componentes principales antes de entrenar los modelos. Tanto este gráfico como los demás han sido generados con nodos de Knime, en este caso Linear Correlation, en los anteriores, Pie Chart y Bar Chart.

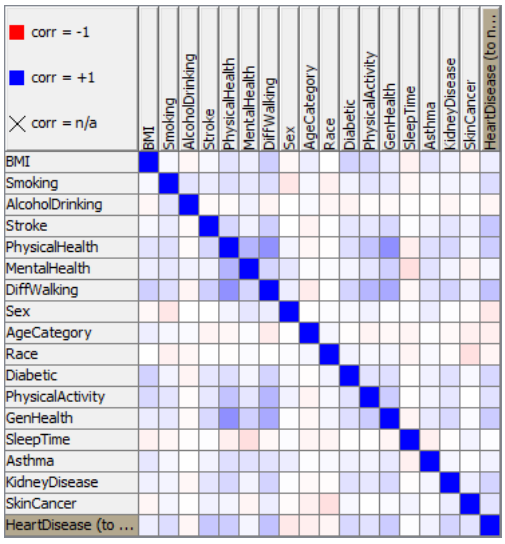


Figura 3: Matriz de correlaciones

1.2. bodyPerformance

Problema de clasificación del estado físico en 4 categorías (multiclase) en función de factores personales y desempeño en distintas prebas físicas, los datos han sido recogidos por la fundación coreana de promoción del deporte. La variable a predecir es la columna class cuyos valores tienen un significado ordinal, de mejor a peor estado físico. En este caso, salvo el sexo, el resto de variabes son numéricas. Las clases estan casi perfectamente balanceadas con un 25 por ciento de valores en cada clase. La clasificación multiclase nos abre la necesidad de encontrar un método de decisión final, de tipo probabilístico, Softmax, o One-Vs-All o generación de clasificadores binarios por pares.

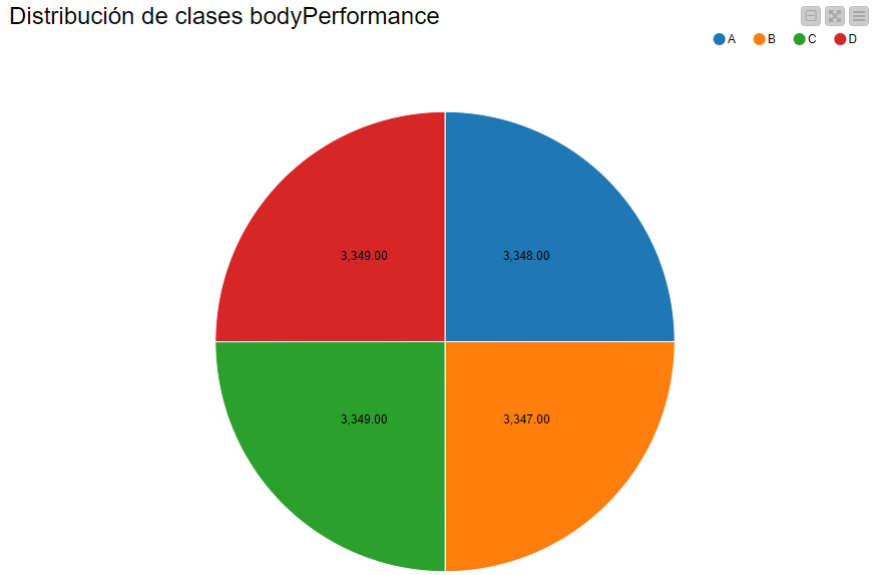


Figura 4: Distribución equilibrada de clases

En la siguiente gráfica mostramos que las variables de aguante de uno de los ejercicios y del índice de masa corporal son monótonas con respecto a las clases a predecir en el orden de forma física, la primera de forma directa y el segundo de forma inversa, la edad en promedio no parece influir por si misma, pero si podría hacerlo en combinación con el resto de las variables.

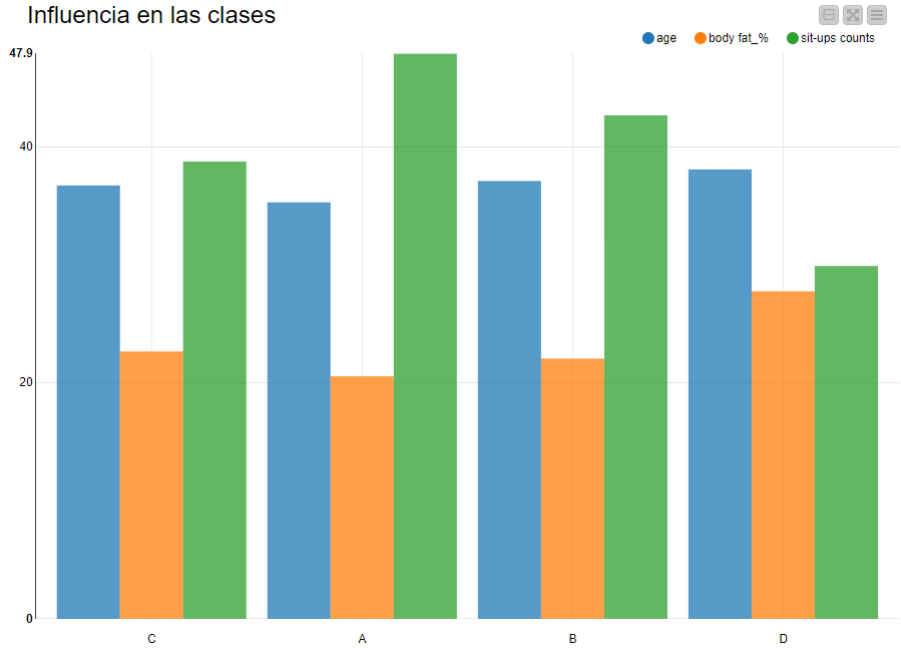


Figura 5: Promedio de variables por clase

En la siguiente matriz de correlaciones se observan múltiples correlaciones altas por lo que es probable que el preprocesado con reducción sea bastante efectivo en este caso, lo que hará que podamos escoger modelos más potentes y hacer que sea factible entrenarlos en tiempo razonable.

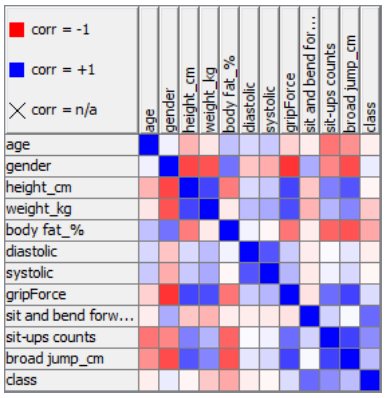


Figura 6: Matriz de correlaciones

1.3. students

Problema de predicción de si un alumno finalizará sus estudios o no, es decir clasificación binaria, esta no es la formulación original del problema, para convertirlo en binario, se han transformado las clases Dropout y Enrolled a Not Graduated, además de este modo logramos un equilibrio importante entre las 2 clases. Datos tomados por el Instituto Politécnico de Portalegre. La variable a predecir es la columna Target. Tenemos variables tanto de tipo numérico, como categóricas y binarias, aunque todas ellas ya en formato numérico. La gran cantidad de variables nos llevará a considerar algún método de selección de características.

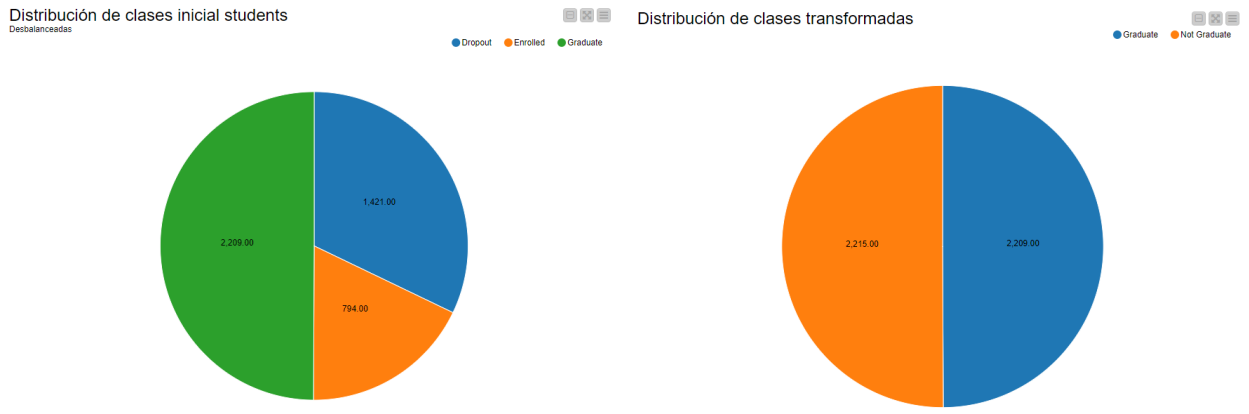


Figura 7: Cambios en la distribución de clases

En la siguiente matriz de correlaciones podemos observar que existen varias variables fuertemente correladas entre sí, por lo que quizá puedan eliminarse en un preprocesado avanzado, por otro lado, también algunas estan altamente correlacionadas con la clase a predecir, lo que puede utilizarse a efectos de interpretaciíon, por ejemplo el género y la edad de comienzo de estudios.

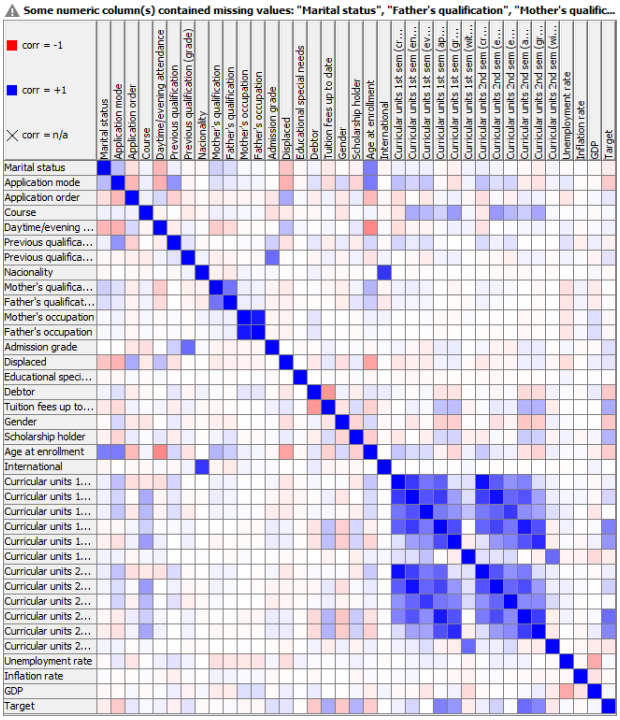


Figura 8: Matriz de correlación de students

2. Procesado de datos

2.1. Básico y obligatorio

En esta sección se describe el preprocesado totalmente esencial para limpieza de los datos y para las características y requisitos de los distintos algoritmos.

2.1.1. heart_2020

En este caso como los datos no tienen valores nulos o faltantes, lo único necesario ha sido cargar los datos mediante el nodo CSV Reader, como los algoritmos usados en este caso son varios de ellos basados en árboles no necesitan conversiones de las variables categóricas.

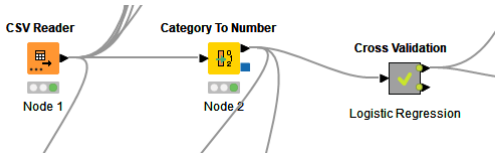


Figura 9: Carga de datos y conversión a variables numéricas

Finalmente, para el modelo de regresión logística se han convertido en numéricas todas las variables categóricas, esto es debido que al ser un modelo lineal, necesita realizar cálculos con los valores, además es recomendable que las variables sean consideradas en el mismo rango para entrenar con mayor eficiencia. Para ello se ha utilizado el objeto Normalizer, respetando la estructura de la validación cruzada y evitando el data snooping se han calculado la media y varianza en los datos de training y se han usado para normalizar tanto los de training como los de test, de este modo los datos de test han quedado lo mas vírgenes posibles y no existen fugas de información que nos hagan sobreestimar los resultados.

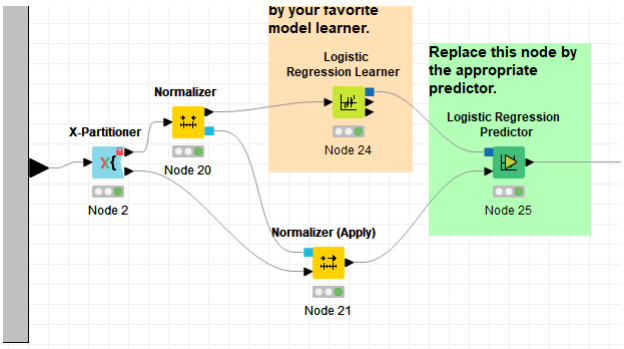


Figura 10: Normalización de las variables

2.1.2. bodyPerformance

En el caso de este dataset, el tratamiento básico es muy similar porque no tiene valores faltantes, luego hacemos la lectura de la misma forma y puesto que la única variable no numérica era género, se ha optado por binarizarla.

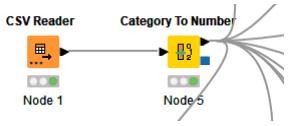


Figura 11: Lectura y binarización

Para los algoritmos en este caso es necesaria la normalización tanto para la máquina de soporte vectorial como para el clasificador de los vecinos más cercanos y la regresión logística.

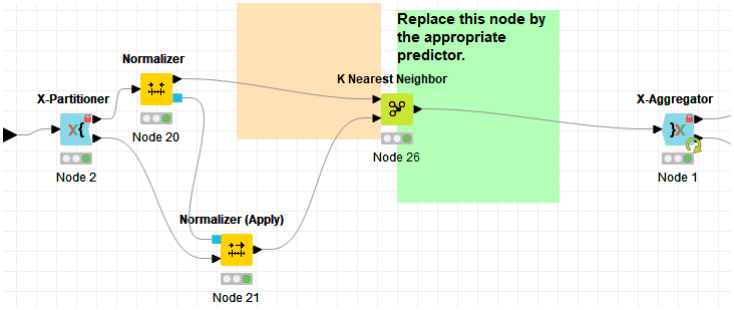


Figura 12: Normalización de las variables

2.1.3. students

Para este conjunto de datos, cargamos los datos y realizamos la transformación de los valores Dropout y Enrolled a Not Graduated en la columna objetivo de la clasificación mediante el nodo String Replace, ha resultado más comodo que usar un diccionario que sería quizá más correcto.

Finalmente en las estadísticas vemos que hay muchos valores perdidos en la columna de estado marital y de ocupaciones de los padres, en el preprocesado avanzado intentaremos buscar una buena solución, en esta sección únicamente dejaremos los valores perdidos en los algoritmos que los admiten, en los que no, se imputaran los valores más comunes, la moda, esta es la mejor opción porque se trata de variables que son códigos de descripciones del estado y de la ocupación que no tienen significado ordinalmente. Para ello usaremos el nodo Missing Values, junto con su paralelo apply para el conjunto de test para no cometer data snooping:

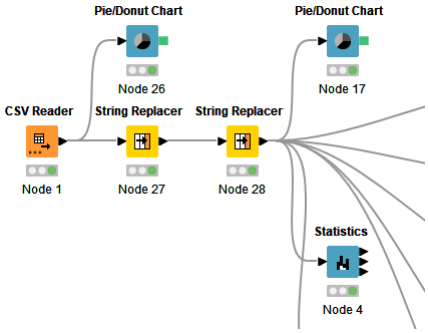


Figura 13: Transformaciones en la columna clase y estadísticas

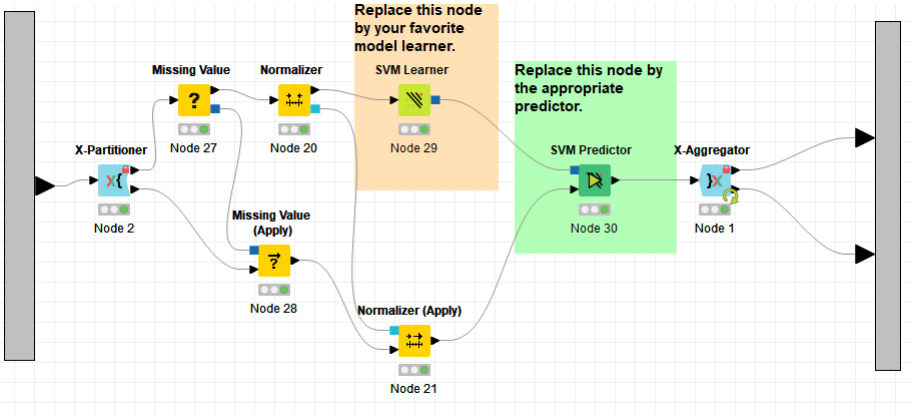


Figura 14: Imputación de valores perdidos

2.2. Avanzado

2.2.1. students

Para el preprocesado avanzado, vamos a llevar a cabo 3 acciones, en primer lugar vamos a imputar valores perdidos en las variables de estado marital y cualificación de padre y madre, se ha comprobado que faltan un cuarto de los valores para estas variables con respecto al número de instancias. Como se tenían suficientes datos en esas variables se ha optado por imputar los valores mediante un clasificador 5-NN de los vecinos más cercano, para ello se ha decidido pasar dichas variables a cadena de texto, porque no tenían significado numérico para la imputación (y además faltan valores) y no usarlas en la predicción.

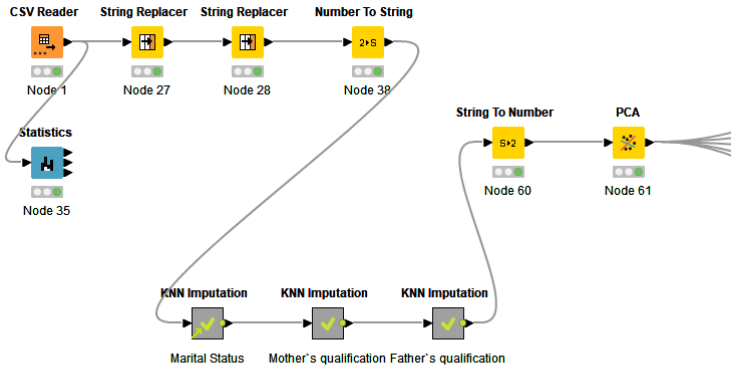


Figura 15: Preprocesado

Para ello se ha creado un metanodo en el que separamos las filas faltantes con el nodo Row Splitter, se ajustan normalizando con respecto a los valores de entrenamiento de este predictor creado para imputar, es decir, los que no tienen el valor perdido. Trás la predicción se usan filtros y nodos de unión de columnas y filas para devolver la forma de partida a los datos pero con los valores correctamente imputados. Se ha optado por llevar a cabo las imputaciones en cadena.

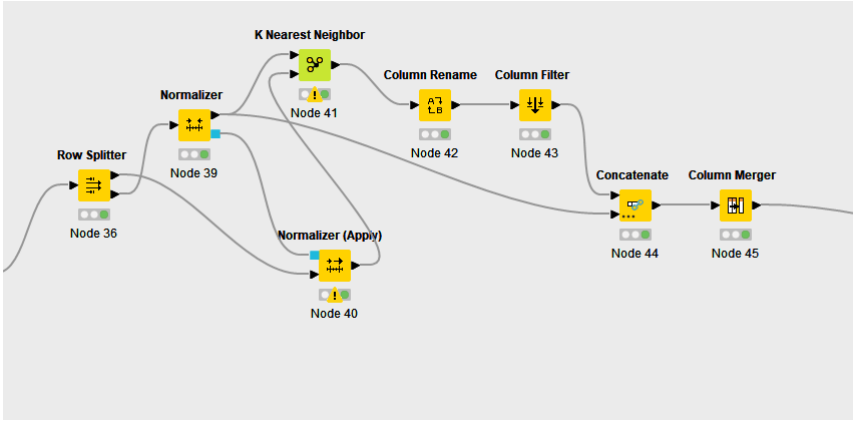


Figura 16: Imputación en base a un predictor 5-NN

Por otro lado, en la matriz de correlaciones que tenemos en el apartado de introducción y análisis exploratorio observamos que teníamos correlaciones altas en 2 grupos de variables y algunas otras con la clase a predecir. Hemos optado entonces por usar la técnica de análisis de componentes principales para reemplazar nuestras variables por las 30 primeras componentes principales, ya que hemos contado las variables con alta correlación y lo hemos restado, este método se basa en diagonalizar la matriz de correlaciones y obtener las combinaciones lineales de variables originales que expliquen un porcentaje de varianza muy alto de la muestra, de este modo reducimos la dimensionalidad manteniendo casi toda la información, esto mejorará los tiempos de ejecución y en algunos algoritmos también su desempeño sobre los datos. Para terminar se ha vuelto a convertir a números y normalizar las variables que hubo que imputar para poder usarlas en los algoritmos que lo requieran.

3. Resultados obtenidos

3.1. heart_2020

En primer lugar vamos a mostrar el flujo realizado para todos los algoritmos seleccionados para este dataset:

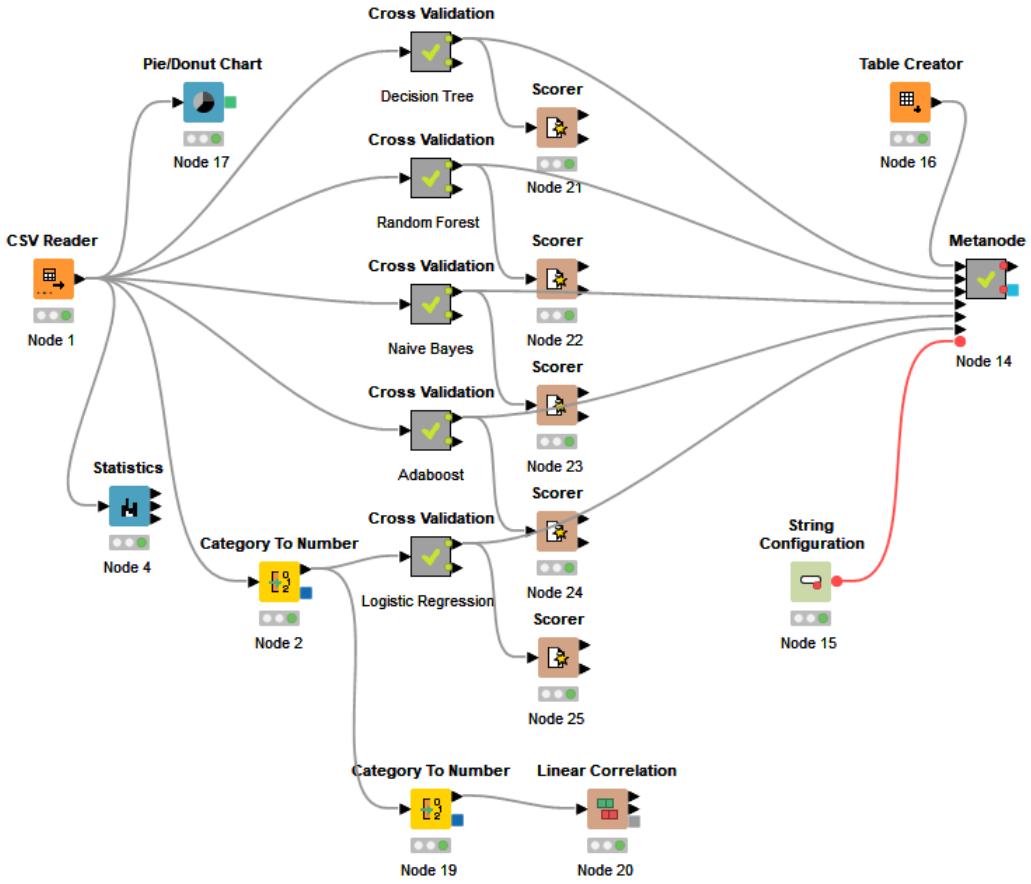


Figura 17: Workflow realizado para el dataset Heart 2020

Para obtener los resultados, basta conectar un objeto scorer a la salida de la validación cruzada, a partir de ahí, se puede obtener la matriz de confusión y el resto de medidas.

3.1.1. Árbol de decisión

Hemos escogido este algoritmo porque en nuestros datos abundan las variables categóricas y es un modelo de gran interpretabilidad en este caso. Una medida de la complejidad del modelo es el número de reglas que se generarían si convirtieramos el árbol es un conjunto de reglas (esto se puede hacer siempre, el recíproco no es cierto) que en este caso es 3505. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C's K	AUC
2110	3874	39989	4733	0.3083	0.3526	0.9117	0.3290	0.8303	0.2323	0.6410

3.1.2. Random Forest

En este caso, la elección de este algoritmo es la más lógica si hemos usado el árbol de decisión por si solo ya que tiene las limitaciones vistas en teoría, la combinación de árboles débiles entrenados con muestras diferentes suele generar un modelo de mayor capacidad predictiva. Una medida de la complejidad del modelo es el número de árboles generados, en este caso 100 y la máxima profundidad que tendrá cada uno que está definida en 10. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C‘s K	AUC
993	682	43181	5850	0.1451	0.5928	0.9845	0.2332	0.8712	0.1902	0.8103

3.1.3. Naive-Bayes

Este algoritmo, pese a tener una suposición de independencia errónea, realiza unos cálculos que suelen resultar en un buen clasificador, si se tratase de usar como estimador, probablemente los resultados serían muy pobres. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C‘s K	AUC
3033	4750	39113	3810	0.4432	0.3897	0.8917	0.4147	0.8312	0.3166	0.8135

3.1.4. ÁdaBoost

Este algoritmo también basado en árboles, se encuentra en el estado del arte y va entrenando árboles teniendo en cuenta los errores de los anteriores e iterando, luego se ha escogido también por la misma razón que RandomForest. Una medida de la complejidad del modelo puede ser el númerop de árboles que se construyen, en este caso 10. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C‘s K	AUC
455	368	43495	6388	0.0665	0.5529	0.9916	0.1187	0.8668	0.0924	0.7696

3.1.5. Regresión logística

El modelo de regresión logística se ha escogido por su simplicidad y velocidad de entrenamiento, además pese a que debemos usar variables numéricas se gana en interpretabilidad del resultado porque se puede obtener una probabilidad de pertenencia a la clase predicha. Una medida de la complejidad del modelo, como se trata de un modelo lineal, puede ser su dimensión de Vapnik Chervonenkis que en este caso es el número de variables más uno, es decir 18. Esto puede usarse para calcular una cota del error de generalización. El esquema del metanodo de validación es muy similar siempre, en este caso así, con normalización: A continuación se muestran la matriz de confusión y las métricas resultantes

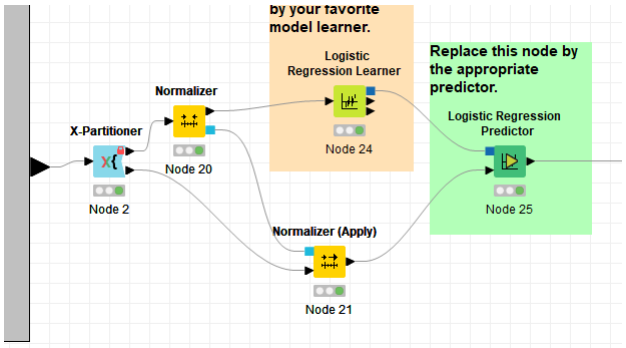


Figura 18: Metanodo de validación cruzada

del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C‘s K	AUC
1084	770	43093	5759	0.1584	0.5847	0.9824	0.2493	0.8712	0.2034	0.7750

3.2. bodyPerformance

En primer lugar vamos a mostrar el flujo realizado para todos los algoritmos seleccionados para este dataset:

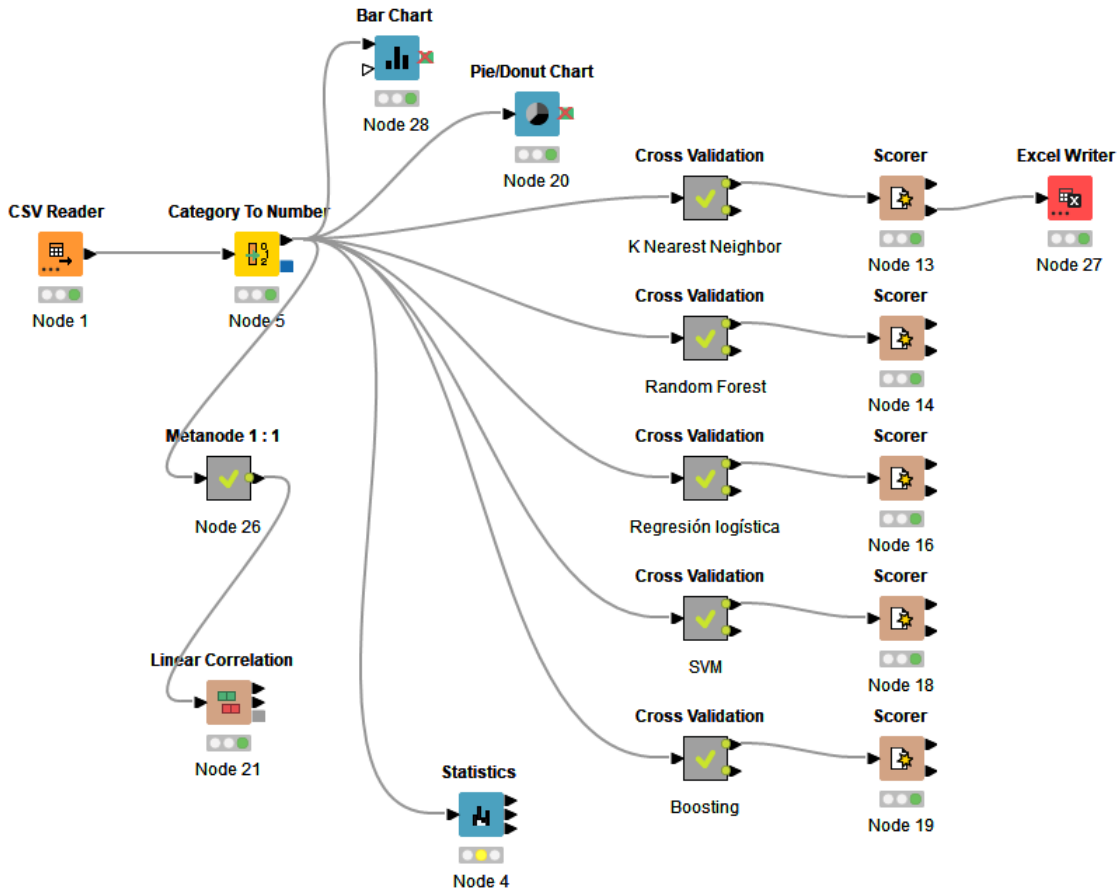


Figura 19: Workflow realizado para el dataset bodyPerformance

Como este conjunto requería clasificación en 4 clases, las métricas habituales deben ser reconsideradas, podemos seguir usando el accuracy y el Cohen's Kappa pero los valores de TPR, TNR, PPV o F1 tienen distinta interpretación, más aun la curva ROC, de hecho podemos calcular una lista de estas métricas considerando una de las clases como positiva y el resto como negativa y interpretarlas a partir de ahí, también existen métricas específicas de multiclasificación más difusas que no consideraremos, nos bastará el accuracy y vamos a considerar que se quiera distinguir al grupo A, la clase con mejor rendimiento del resto y calcular las métricas en función de esta separación, se podría hacer lo mismo para medir como distingue el algoritmo alguna de las demás del resto. Además se podría calcular AUC en función de la elección de la clase A como positiva y el resto como negativa y se podría mostrar la gráfica pero no nos parece muy representativa del modelo general, habría que representar 4 gráficas y analizarlas en conjunto.

Además la naturaleza ordenada de las clases podría llevarnos a plantear el problema como regresión y generar nuestra propia métrica que comparara la estimación de un modelo de regresión que predijera entre 0 y 1 y establecer rangos en los que encuadrar las instancias en las 4 clases posibles y calcular el desempeño del modelo a partir de esos datos.

Pasamos ahora a mostrar las métricas obtenidas y la justificación de haber escogido estos modelos:

3.2.1. K-Nearest Neighbor

En este caso hemos considerado esta elección porque casi todas las variables son numéricas puras sin transformar. Hemos tenido que usar normalización para que no influyan las distintas escalas en el funcionamiento del algoritmo. En este caso una mediad de complejidad es el número de vecinos a considerar, la K, el valor usado ha sido 7. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

class \ Clas...	A	B	C	D
A	2558	687	97	6
B	1042	1575	631	99
C	344	1028	1787	190
D	73	276	761	2239

Correct classified: 8.159 Wrong classified: 5.234

Accuracy: 60,92% Error: 39,08%

Cohen's kappa (κ): 0,479%

Figura 20: Matriz de confusión de 7-NN

TPR	PPV	TNR	F1	%class	C's K
0.7640	0.6368	0.8548	0.6946	0.6092	0.4789

3.2.2. Random Forest

Se ha optado por este algoritmo para usarlo junto al de boosting como representante de los algoritmos construidos con árboles de decisión, frente al resto de algoritmos que son puramente ajuste de pesos basados en optimización numérica. Una medida de la complejidad del modelo es el número de árboles generados, en este caso 100 y la máxima profundidad que tendrá cada uno que está definida en 10. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

class \ Pre...	A	B	C	D
A	2895	411	32	10
B	797	2031	403	116
C	287	648	2235	179
D	42	165	411	2731

Correct classified: 9.892 Wrong classified: 3.501

Accuracy: 73,859% Error: 26,141%

Cohen's kappa (κ): 0,651%

Figura 21: Matriz de confusión de Random Forest

TPR	PPV	TNR	F1	%class	C's K
0.8646	0.7199	0.8879	0.7857	0.7385	0.6514

3.2.3. Regresión logística

Usaremos el algoritmo con el optimizador de gradiente descendente estocástico, se usa Softmax para efectuar la clasificación multiclase. Una medida de la complejidad del modelo, como se trata de un modelo lineal, puede ser su dimensión de Vapnik Chervonenkis que en este caso es el número de variables más uno, es decir 12. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

class \ Pre...	A	B	C	D
A	2474	811	63	0
B	795	1485	959	108
C	264	727	1739	619
D	43	179	556	2571

Correct classified: 8.269 Wrong classified: 5.124

Accuracy: 61,741% Error: 38,259%

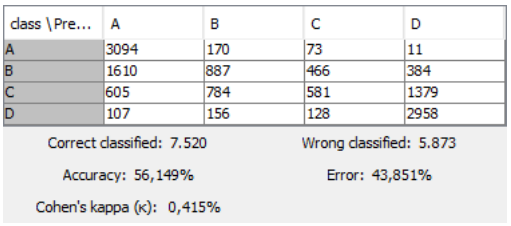
Cohen's kappa (κ): 0,49%

Figura 22: Matriz de confusión de Regresión Logística

TPR	PPV	TNR	F1	%class	C's K
0.7389	0.6918	0.8903	0.7146	0.6174	0.4899

3.2.4. Máquina de soporte vectorial

Este algoritmo se ha utilizado para tratar de comprobar si efectivamente tiene problemas para el manejo de problemas multiclase, hemos probado varios kernels, y comprobado la lentitud de la implementación de KNIME como se advierte en la documentación del algoritmo, a pesar de todos es planteable, pues su entrenamiento supuso unos 5 minutos, en el problema heart es inviable con el gran tamaño del dataset. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:



class \Pre...	A	B	C	D
A	3094	170	73	11
B	1610	887	466	384
C	605	784	581	1379
D	107	156	128	2958

Correct classified: 7.520 Wrong classified: 5.873

Accuracy: 56,149% Error: 43,851%

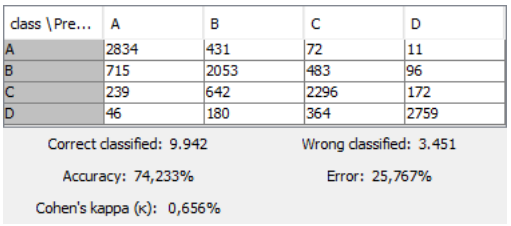
Cohen's kappa (κ): 0,415%

Figura 23: Matriz de confusión de Support Vector Machine

TPR	PPV	TNR	F1	%class	C‘s K
0.4397	0.4176	0.7956	0.4283	0.3647	0.1529

3.2.5. Boosting

En este caso no hemos podido usar AdaBoost pues está implementado para clasificación binaria en Weka, hemos optado entonces por usar el algoritmo Gradient Boosting propio de Knime. Una medida de la complejidad del modelo puede ser el númerop de árboles que se construyen, en este caso 500, junto con el máximo de profundidad, 4. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:



class \Pre...	A	B	C	D
A	2834	431	72	11
B	715	2053	483	96
C	239	642	2296	172
D	46	180	364	2759

Correct classified: 9.942 Wrong classified: 3.451

Accuracy: 74,233% Error: 25,767%

Cohen's kappa (κ): 0,656%

Figura 24: Matriz de confusión de Gradient Boosting

TPR	PPV	TNR	F1	%class	C‘s K
0.8465	0.7392	0.9004	0.7892	0.7423	0.6564

3.3. students

De nuevo comenzamos mostrando el flujo básico para este algoritmo. En la sección de configuración se estudiará una versión más compleja de los algoritmos buscando los mejores hiperparámetros y se aplicará el preprocesado avanzado descrito en la sección anterior.

3.3.1. Máquina de soporte vectorial

En este caso como tenemos un número de instancias más reducido, podemos optar por usar la potencia de este algoritmo porque los tiempos de ejecución son más razonable incluso en la validación cruzada con 5 particiones. En la sección avanzada se tratará de probar otros kernels. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C‘s K	AUC
1937	430	1785	272	0.8768	0.8183	0.8058	0.8465	0.8413	0.6826	0.9196

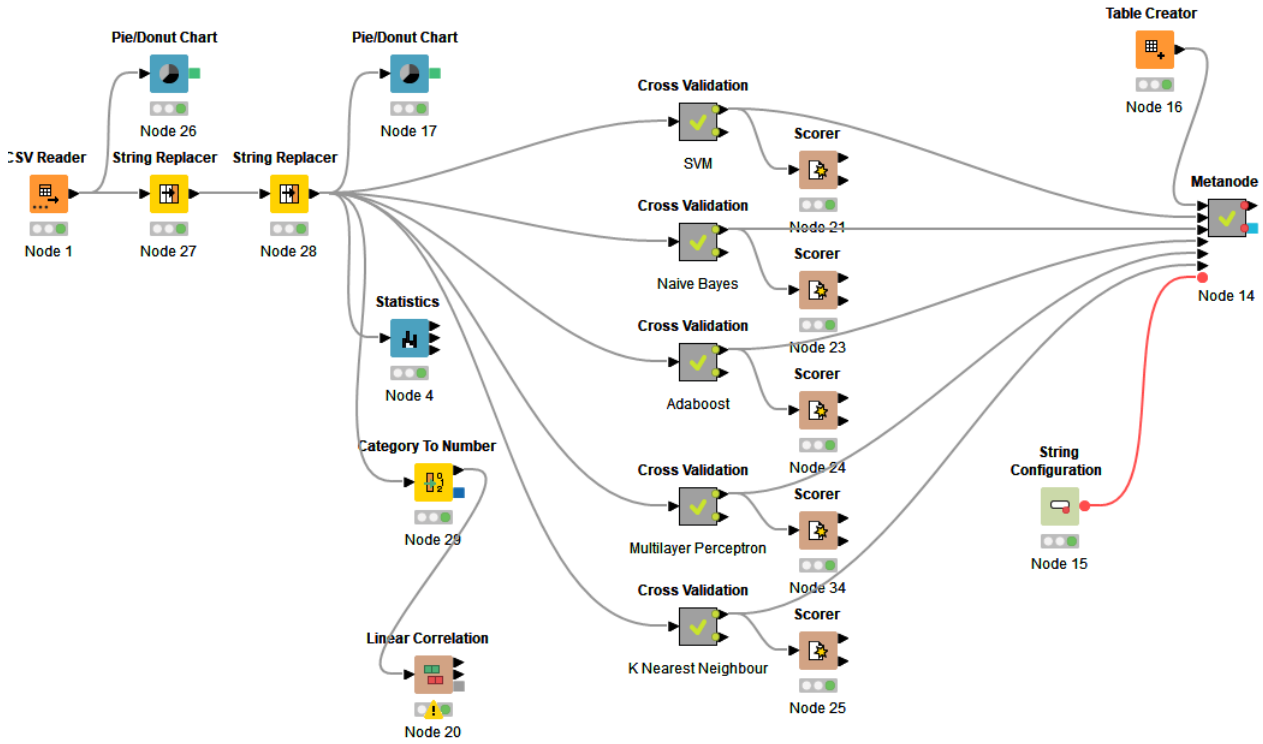


Figura 25: Workflow realizado para el dataset students en la versión básica

3.3.2. Naive Bayes

Usaremos Naive Bayes como clasificador base para realizar la comparación porque suele obtener métricas razonables, aunque, es de esperar que el resto de algoritmos más avanzados y con mayor complejidad paramétrica lo superen. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C's K	AUC
2038	955	1260	171	0.9226	0.6809	0.5688	0.7835	0.7455	0.4912	0.8513

3.3.3. Adaboost

Como representante de los algoritmos de ensembling, usaremos el AdaBoost que puede ser indicado por la gran variedad de características que tenemos en el dataset. Una medida de la complejidad del modelo puede ser el númerop de árboles que se construyen, en este caso 10. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C's K	AUC
1921	522	1693	288	0.8696	0.7863	0.7643	0.8259	0.8169	0.6339	0.8843

3.3.4. Multilayer Perceptron

Usaremos esta red neuronal básica con 2 capas ocultas de 10 neuronas para tratar de ajustar un modelo que generalice correctamente y obtenga alta precisión en las métricas. Se usaran 100 iteraciones del algoritmo de descenso de gradiente predeterminado. Una medida de la complejidad del modelo puede ser el número efectivo de parámetros que se ajustan en el entrenamiento, en este caso dada la arquitectura de la red son 491 pesos. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C's K	AUC
1872	453	1762	337	0.8474	0.8052	0.7955	0.8258	0.8214	0.6429	0.8962

3.3.5. K Nearest Neighbor

La naturaleza numérica de los datos nos hace considerar este algoritmo, además su base de instancias pasadas hace que las predicciones tengan fácil interpretación. En este caso una mediad de complejidad es el número de vecinos a considerar, la K, el valor usado ha sido 7. A continuación se muestran la matriz de confusión y las métricas resultantes del proceso de validación y test:

TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C's K	AUC
1924	667	1548	285	0.8710	0.7426	0.6989	0.8017	0.7848	0.5697	0.8531

4. Configuración de algoritmos

Para el conjunto de datos students, estudiaremos en esta sección una elección de hiperparámetros que mejore los resultados con el preprocesado avanzado que se explicó anteriormente.

- En el algoritmo Support Vector Machine la implementación de Knime permite cambiar el Kernel por uno hipertangente y otro de base radial, se ha testeado la ejecución del algoritmo con ellos y aunque algunos consiguen reducir mucho los errores de uno de los tipos, lo hacen a costa de aumentar muchísimo el otro, de modo que salvo para aplicaciones muy concretas se hace inasumible su utilización, no es el caso, por tanto se mantiene el Kernel polinomial habitual que es uno de los mejores modelos kigrados, además pese haber reducido las características en el preprocesado las métricas mejoran como se estudiará en la siguiente sección.
- En el algoritmo AdaBoost se ha modificado el número de iteraciones en el entrenamiento y mejora de los clasificadores débiles, ha ocurrido un fenómeno curioso y similar al sobreentrenamiento de las redes neuronales, existe una mejora al aumentar las iteraciones pero a partir de cierto punto óptimo se conmienza a empeorar, por ello se ha optado por utilizar 100 iteraciones. También se ha observado que al activar la opción de volver a muestrear los datos en el entrenamiento, los resultados mejoran.
- En la red neuronal MLP se han probado arquitecturas con diversos número de capas y de neuronas en cada una de ellas, la implementación sólo permite redes totalmente conectadas y no sabemos si el optimizador usa Dropout a la hora de entrenar, luego se ha probado arbitrariamente, en Python con scikit-learn se podrían hacer búsquedas de hiperparámetros más intensivas con la función GridSearch. Finalmente se ha obtenido que la red más pequeña a partir de la cual no mejoran los resultados si la aumentamos es una con 3 capas ocultas además de la de entrada y salida con 12 neuronas por capa. De nuevo para evitar el sobreentrenamiento se han optimizado las iteraciones y con 100 se obtiene un buen resultado. El fenómeno es el típico en redes neuronales, una curva que disminuye el error pero a partir de cierto punto aumenta.

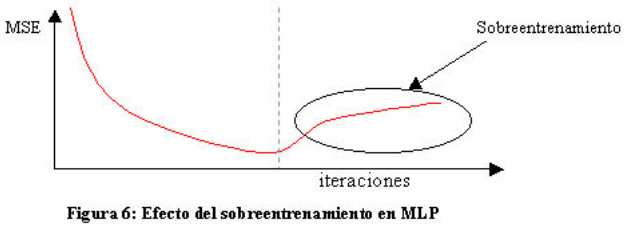


Figura 26: Sobreentrenamiento típico en redes neuronales

- Finalmente en K Nearest Neighbor se han probado diferentes números de vecinos para estimar, es decir la K, se muestra la siguiente gráfica comparativa, se ha escogido finalmente 8 vecinos y el método de desempate que implementa Knime está basado en el primer vecino.

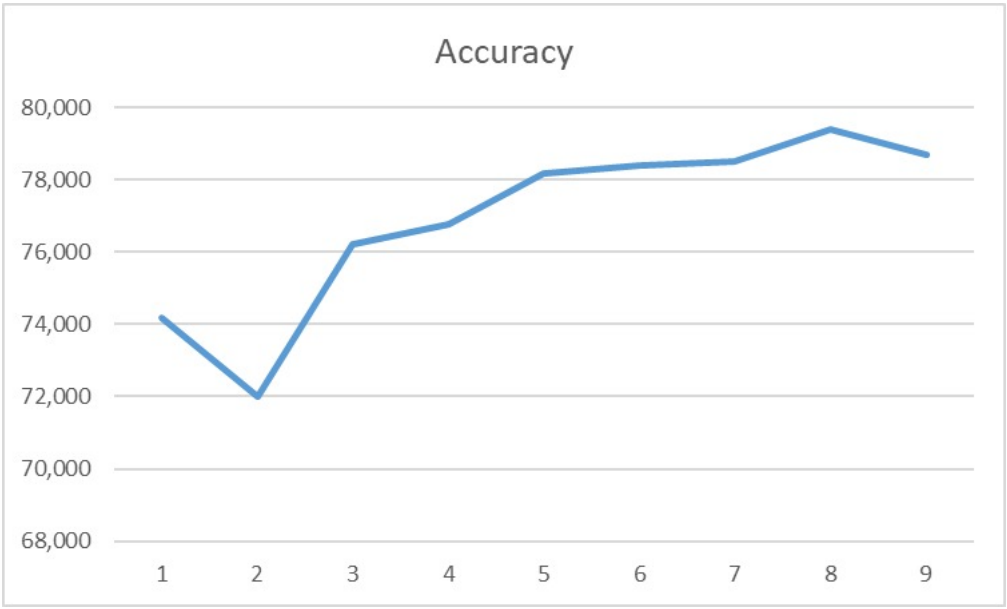


Figura 27: Gráfica de asistencia para elegir K

Se compararan los resultados en la siguiente sección pero la tónica general es que se mejora muy ligeramente el resultado, pero teniendo en cuenta el preprocesado y la selección de variables tan intensiva que se ha usado, hemos conseguido una mejora importante en tiempos y también en algunos casos rendimiento en métricas específicas como TPR o PPV.

5. Análisis de resultados

Llevaremos a cabo ahora el análisis de los resultados obtenidos en los algoritmos de forma comparativa, para ello se usaran tablas con las métricas y las curvas ROC en los casos de clasificación binaria, para esto último se han usado múltiples nodos de filtrado y se ha seguido el tutorial a tal efecto de la página web de Jorge Casillas.

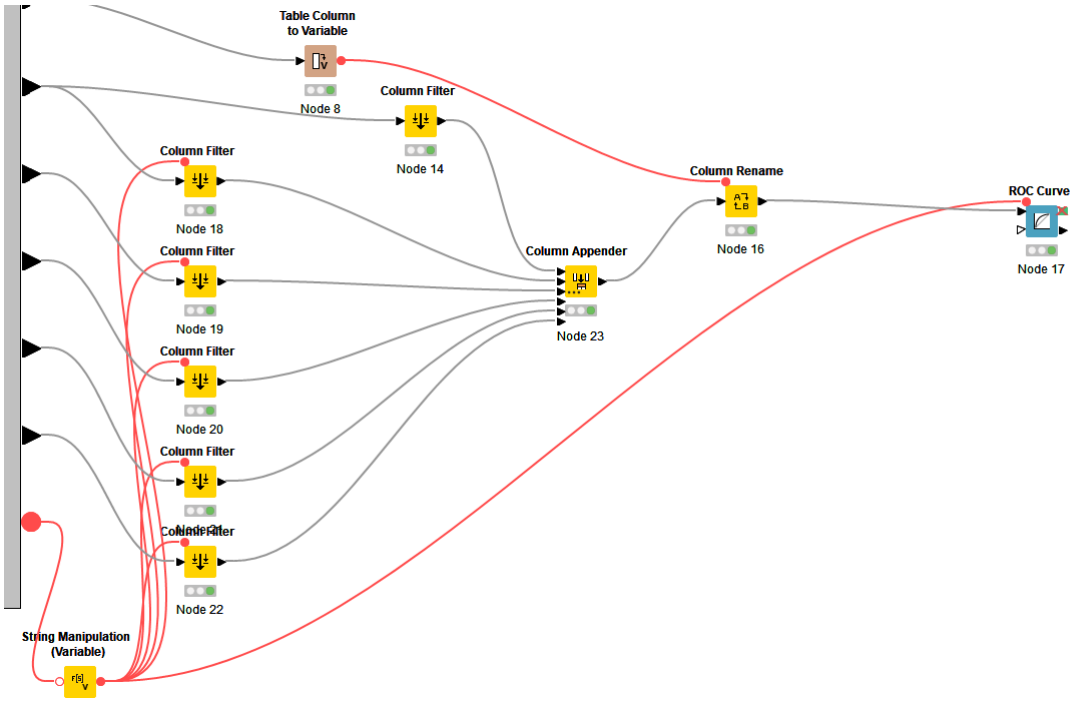


Figura 28: Metanodo de extracción de curvas ROC

5.1. heart_2020

En este dataset recordemos que las clases estaban desbalanceadas en torno a un 85 por ciento hacia la clase negativa, es por tanto que el accuracy no será una métrica a tener muy en cuenta ya que un clasificador que escoja siempre la clase mayoritaria conseguirá un "buenresultado y sin embargo será totalmente inútil. Por eso en este caso otras métricas como el Recall o el área bajo la curva serán capaces de medir mejor la eficacia de los algoritmos.

Heart 2020	TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C's K	AUC
Decision Tree	2110	3874	39989	4733	0.3083	0.3526	0.9117	0.3290	0.8303	0.2323	0.6410
Random Forest	993	682	43181	5850	0.1451	0.5928	0.9845	0.2332	0.8712	0.1902	0.8103
Naive Bayes	3033	4750	39113	3810	0.4432	0.3897	0.8917	0.4147	0.8312	0.3166	0.8135
Adaboost	455	368	43495	6388	0.0665	0.5529	0.9916	0.1187	0.8668	0.0924	0.7696
R. Logística	1084	770	43093	5759	0.1584	0.5847	0.9824	0.2493	0.8712	0.2034	0.7750

Para comenzar el análisis estamos de acuerdo en que una propiedad que tendrá un buen algoritmo para este conjunto de datos será que en la matriz de confusión tengamos un bajo número de falsos negativo, es decir de personas que no sean diagnosticadas y finalmente si que tengan el fallo. En este sentido, el algoritmo Naive Bayes consigue con cierta amplitud esta propiedad con respecto a los demás, pese a eso su TNR es baja en comparación con el resto ya que en el caso de los errores del otro tipo (falsos positivos) destaca como el que más comete. Es decir es el algoritmo que menos futuros enfermos deja de diagnosticar pero identifica incorrectamente bastantes como enfermos algunos que no lo serán. Por otro lado, también destaca con respecto a los demás en los verdaderos futuros enfermos detectados, es el que más detecta con diferencia, su TPR también es el mejor de todos. La curva ROC y su área también detectan la bondad en este sentido consiguiendo una buena puntación, lo mismo ocurre con la métrica F1 que suele usarse mucho en casos de clases desbalanceadas. Otro modelo que podríamos plantearnos como asumible sería el árbol de decisión como segundo mejor F1 score, pero estaríamos reduciendo los falsos positivos a costa de aumentar los falsos negativos, lo que para esta aplicación no es asumible.

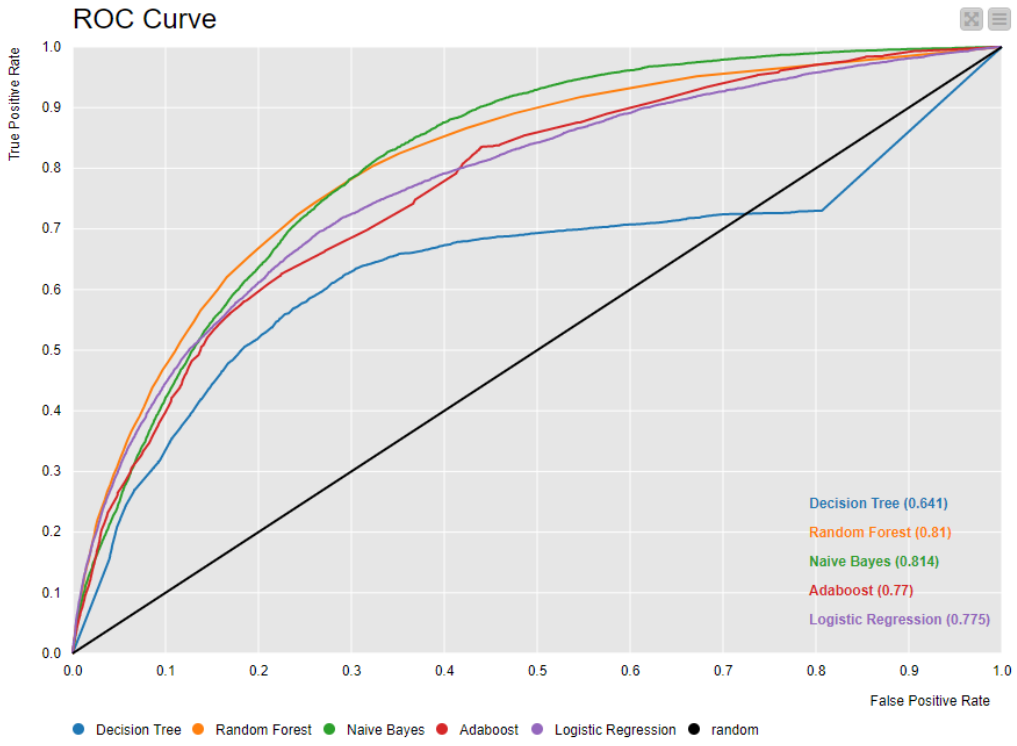


Figura 29: Curvas ROC para el dataset heart 2020

Es por ello que se decide como el mejor modelo el clasificador Naive Bayes, se da la paradoja de que es el segundo modelo con peor accuracy, pero a pesar de todo por la importancia de los tipos de errores para este problema es la mejor elección de los considerados. Sin embargo, parece que los resultados son bastante pobres, aunque tendrían cierta utilidad en la detección de futuros casos. Debería considerarse utilizar técnicas de selección de instancias para disminuir la clase mayoritaria (undersampling) o generación de instancias artificiales de la clase minoritaria como SMOTE.

5.2. bodyPerformance

Recordemos que como este conjunto requería clasificación en 4 clases, las métricas habituales deben ser reconsideradas, podemos seguir usando el accuracy pero para los valores de TPR, TNR, PPV o F1 hemos escogido un criterio, vamos a considerar que se quiere distinguir al grupo A, la clase con mejor rendimiento del resto y calcular las métricas en función de esta separación. Pueden ser más interesantes en este caso volver a analizar las matrices de confusión porque en este caso deberían ser muy informativas.

BodyPerformance	TPR	PPV	TNR	F1	%class	C's K
7-NN	0.7640	0.6368	0.8548	0.6946	0.6092	0.4789
Random Forest	0.8647	0.7200	0.8879	0.7857	0.7386	0.6515
Regresión logística	0.7389	0.6918	0.8903	0.7146	0.6174	0.4899
SVM	0.4397	0.4176	0.7956	0.4283	0.3647	0.1529
Boosting	0.8465	0.7392	0.9004	0.7892	0.7423	0.6564

Descartamos los resultados de Support Vector Machine por ser muy malos, incluso pareciera que pudiera haber algún error en la ejecución del algoritmo. Es posible que sea debido a que la adaptación a problemas multiclase, que es uno contra todos en la implementación de Knime, tenga problemas en este caso en el que existe una relación de orden entre las clases. Volvamos a mostrar las matrices de confusión del resto para analizarlas conjuntamente con el resto de las métricas, estan en el mismo orden que en la tabla, se puede comprobar el accuracy:

class \ Clas...	A	B	C	D
A	2558	687	97	6
B	1042	1575	631	99
C	344	1028	1787	190
D	73	276	761	2239

Correct classified: 8.159 Wrong classified: 5.234

Accuracy: 60,92% Error: 39,08%

Cohen's kappa (κ): 0,479%

class \ Pre...	A	B	C	D
A	2895	411	32	10
B	797	2031	403	116
C	287	648	2235	179
D	42	165	411	2731

Correct classified: 9.892 Wrong classified: 3.501

Accuracy: 73,859% Error: 26,141%

Cohen's kappa (κ): 0,651%

class \ Pre...	A	B	C	D
A	2474	811	63	0
B	795	1485	959	108
C	264	727	1739	619
D	43	179	556	2571

Correct classified: 8.269 Wrong classified: 5.124

Accuracy: 61,741% Error: 38,259%

Cohen's kappa (κ): 0,49%

class \Pre...	A	B	C	D
A	2834	431	72	11
B	715	2053	483	96
C	239	642	2296	172
D	46	180	364	2759

Correct classified: 9.942 Wrong classified: 3.451

Accuracy: 74,233% Error: 25,767%

Cohen's kappa (κ): 0,656%

Un buen punto de partida para el análisis puede ser darse cuenta que el accuracy (este si está calculado con todas las clases, no particularizado a A) es bastante mejor en los algortimos de ensemble learning, tanto Random Forest como Boosting, esto era de esperar porque manejan muy bien la clasificación no binaria. Por otro lado, una apreciación general inspirada por la propia naturaleza ordinal de las clases del problema es que los 4 algoritmos consiguen hacer una distinción bastante correcta de las clases polos opuestos, es decir, muy pocas veces confunden alguien con muy buena forma física con todo lo contrario, esto se aprecia en que el primer y último valor de la antidiagonal son muy bajos. Aprovechando este análisis, se puede observar que para los ensembles, los otros 2 valores de la antidiagonal son más bajos que en los otros 2 algoritmos, es decir, los errores entre las clases intermedias B y C.

Respecto a elegir al mejor algoritmo para el tipo de análisis que hemos elegido basta mirar la tabla, el algoritmo de Boosting es el mejor en todas las métricas salvo en TPR donde es muy similar a Random Forest. Esto junto al accuracy y a lo observado en las matrices nos hace escoger Boosting muy seguido de Random Forest. La valoración de los resultados es bastante positiva, pues como hemos indicado la naturaleza de este problema hace aceptable que si bien no se tiene una accuracy perfecta, no se cometen errores graves casi nunca.

5.3. students

Para este problema tenemos que comparar todos los pares de algoritmos entre la versión por defecto y la preprocesada y configurada, finalmente se hará la elección del mejor modelo, en este caso, pese a que haremos algunas consideraciones respecto al resto de métricas el accuracy será muy buen indicador salvo que se quiera incidir muy claramente en un criterio que penalice mucho un tipo de error frente al otro, algo que no parece indicarse claramente en la naturaleza del problema.

students	TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C's K	AUC
Support Vector Machine	1937	430	1785	272	0.8769	0.8183	0.8059	0.8466	0.8413	0.6827	0.9197
Naive Bayes	2038	955	1260	171	0.9226	0.6809	0.5688	0.7835	0.7455	0.4912	0.8513
AdaBoost	1921	522	1693	288	0.8696	0.7863	0.7643	0.8259	0.8169	0.6339	0.8843
Multilayer Perceptron	1872	453	1762	337	0.8474	0.8052	0.7955	0.8258	0.8214	0.6429	0.8962
7 Nearest Neighbor	1924	667	1548	285	0.8710	0.7426	0.6989	0.8017	0.7848	0.5697	0.8531

A continuación mostramos los resultados correspondientes a la versión mejorada:

students improved	TP	FP	TN	FN	TPR	PPV	TNR	F1	%class	C's K	AUC
Support Vector Machine	1930	425	1790	279	0.8737	0.8195	0.8081	0.8457	0.8409	0.6818	0.9191
AdaBoost	1877	476	1739	332	0.8497	0.7977	0.7851	0.8229	0.8174	0.6348	0.8888
Multilayer Perceptron	1961	472	1743	248	0.8877	0.8060	0.7869	0.8449	0.8373	0.6745	0.9064
8 Nearest Neighbor	1944	686	1529	265	0.8800	0.7392	0.6903	0.8035	0.7850	0.5702	0.8563
Naive Bayes	1907	754	1461	302	0.8633	0.7166	0.6596	0.7832	0.7613	0.5227	0.8435

- Con respecto a SVM, como comentamos en otras secciones, no se ha logrado una configuración que mejore los resultados de la por defecto, observamos que el efecto del preprocesado, ha reducido bastante la dimensión del problema y los resultados son prácticamente calcados que en la otra versión.
- En el caso de Naive Bayes ocurre lo mismo, es reseñable que sin embargo con la imputación realizada en el preprocesado, el algoritmo obtiene mejores resultados en accuracy y un F1 score muy similar, este algoritmo podría haber sido sensible a una mala imputación y no ha sido el caso.
- Para el algoritmo AdaBoost si que se realizaron cambios reseñables, observamos que como es resistente al sobreajuste el aumentar las iteraciones ha mejorado el modelo, tanto el AUC como la accuracy son ligeramente mejores a la versión estándar.
- El Perceptron multicapa también ha respondido mejor a los cambios, hemos aumentado la capacidad del modelo agregando una capa más y 2 neuronas más por capa, además la eliminación de variables correladas mediante el análisis de componentes principales suele mejorar la convergencia del algoritmo de optimización. De esta forma hemos intentado usar un número de de iteraciones no muy alto para evitar el sobreentrenamiento.
- El clasificador del vecino más cercano ha sufrido pocos cambios con respecto a configuración, sólo se aumentó K en 1 por lo que prácticamente los cambios se deben al preprocesado, de nuevo un algoritmo que podría haber respondido mal a la imputación no lo ha hecho y obtiene métricas muy parecidas a su predecesor.

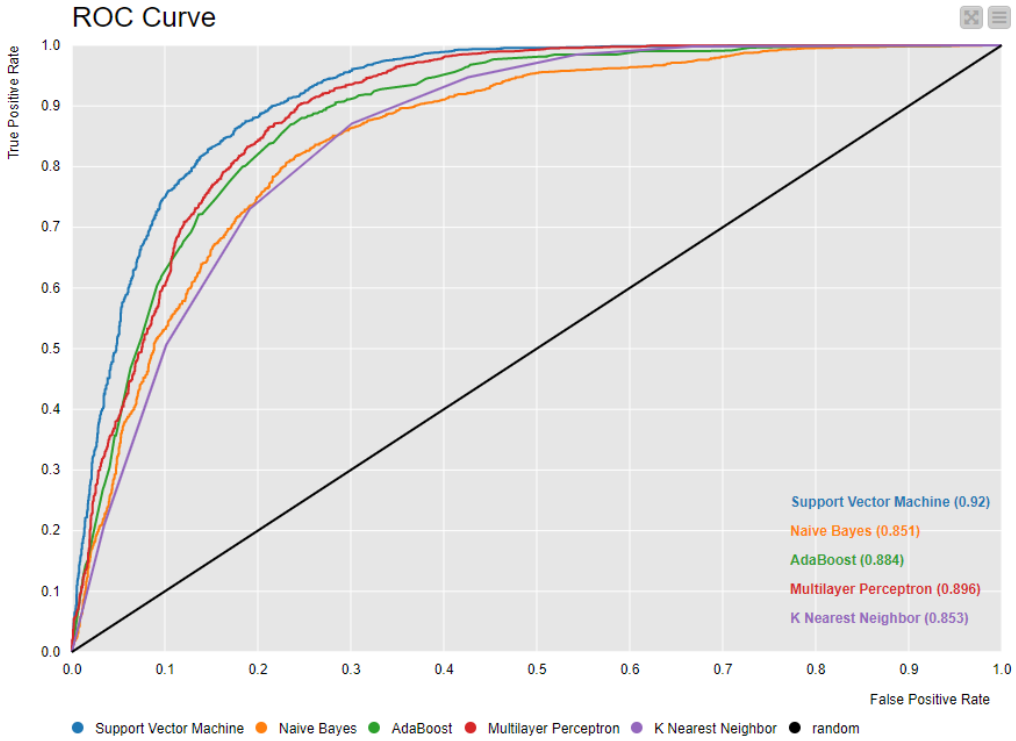


Figura 30: Curvas ROC para el dataset heart 2020

Finalmente a la hora de escoger el mejor modelo, vamos a fijarnos principalmente en el accuracy y el F1 score, en este sentido el mejor es la Support Vector Machine sin preprocesado seguida de la versión con preprocesado muy de cerca. De hecho nuestra elección será la elección con él, esto es debido a que si tuvieramos que entrenar un modelo final con una cantidad más masiva de datos, o tenemos valores perdidos, o tenemos que reentrenar muy habitualmente, el algoritmo es más rápido si realizamos una alta reducción de dimensionalidad como en nuestro preprocesado.

6. Interpretación de los datos

La gran ventaja de los árboles de decisión es su alta interpretabilidad, se puede saber exactamente el criterio por el que se toma la decisión en la clasificación, en esta sección vamos a realizar un pequeño análisis con la ayuda del nodo interactivo Decision Tree to Image. Nos limitaremos por motivos de espacio, simplicidad y tiempo al dataset heart.

6.1. heart_2020

Observamos que trás el entrenamiento, la primera variable que se considera para dividir es la salud general lo cual parece bastante razonable, está dividida en 5 grupos en los cuales, la mayoría de la clase positiva está en el grupo de peor salud.

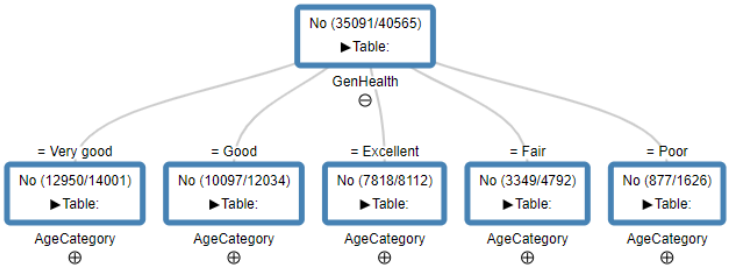


Figura 31: Primer nivel del árbol

Si nos centramos en dicho grupo, podemos ver que es determinante el grupo de edad, algo también muy razonable, detectando los grupos a partir de 65 años como los problemáticos.

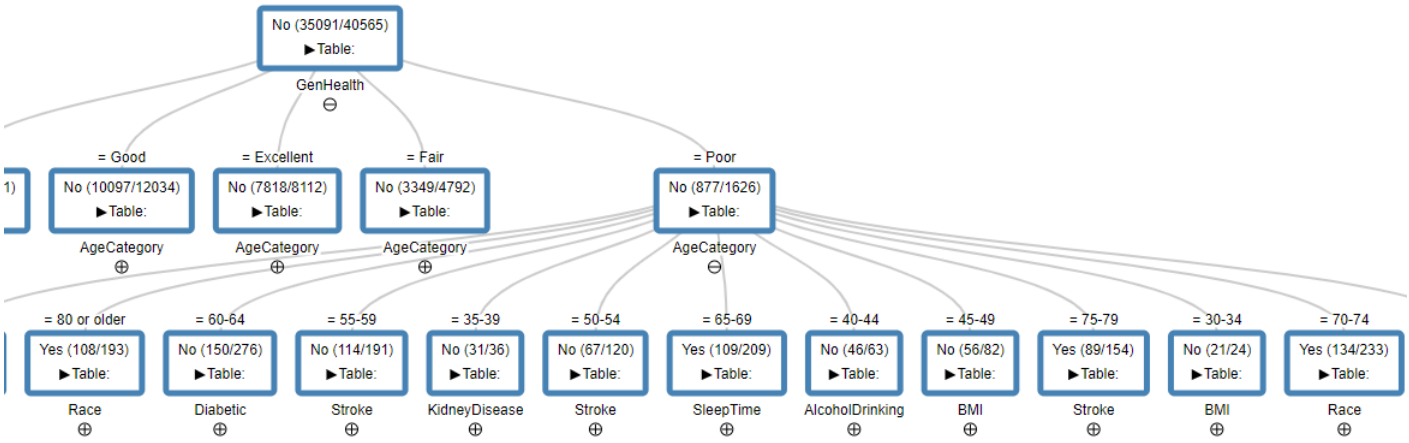


Figura 32: Segundo nivel del árbol

Dentro de los distintos grupos de edad, el algoritmo ha generado los nodos teniendo en cuenta la experiencia pasada del entrenamiento. Por ejemplo en los grupos de mayor edad pregunta por la raza del sujeto, mientras en grupos de mediana edad como el de 50-54 años pregunta por un ataque al corazón pasado, lo cuál se sabe en la actualidad que es muy determinante para este grupo de edad. En el caso de los grupos de menor directamente determina que no lo tendrán. Se puede observar que en aquellos con un ataque pasado, aumenta su posibilidad de tener otro y pregunta de nuevo por la raza como hacía con la gente más mayor.

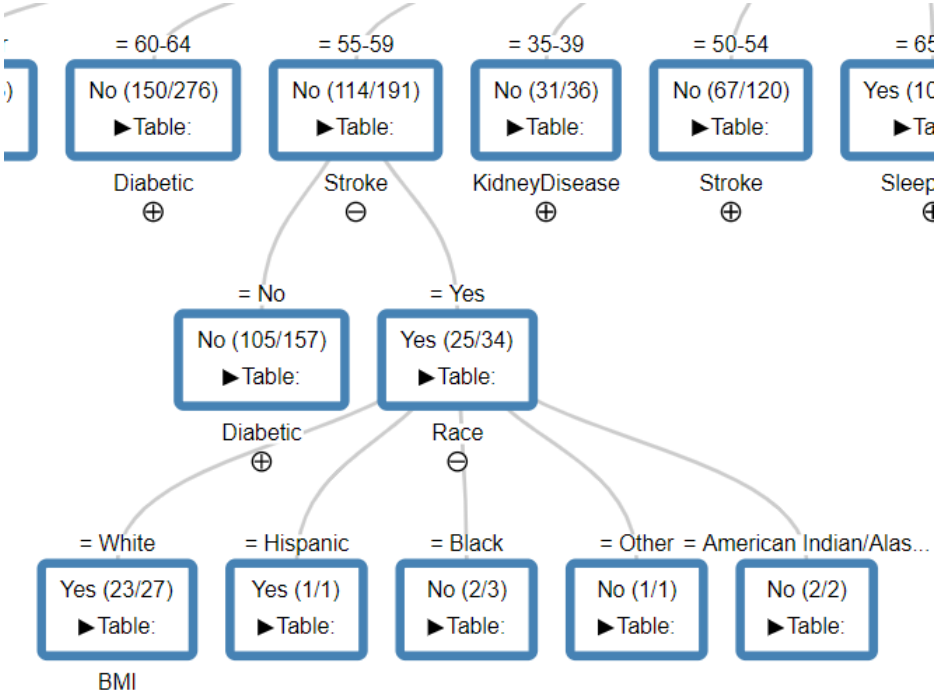


Figura 33: Grupo de edad 50-54

Finalmente si vamos por la rama de gente con muy buena salud general y avanzamos un par de ramas en los hombres de más de 80 años, se observa que pregunta por la raza y en el caso de asiáticos lo clasifica como futuro infarto pese a que en todo el resto de razas respondería que no. Esto es debido a que sólo disponía de un individuo de entrenamiento en dichas condiciones y por tanto lo clasifica así por el criterio de construcción del árbol.

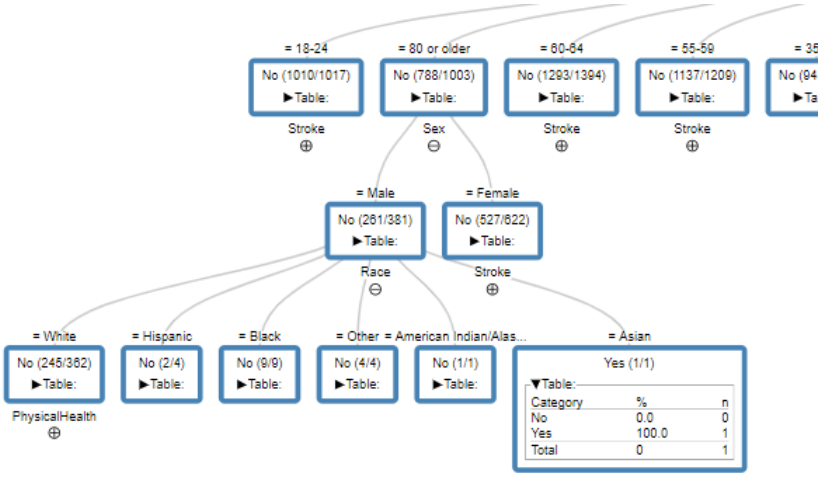


Figura 34: Rama de buena salud general

Esto nos recuerda que por muy bueno que pueda ser un algoritmo, no existe ninguno universal y que lo fundamental no son tanto los algoritmos sino la calidad, cantidad, variedad y autenticidad de los datos.

7. Contenido adicional

7.1. Descripción de los algoritmos usados

7.1.1. K-Nearest Neighbours

El algoritmo de los vecinos mas cercanos se encuentra en el paradigma de los modelos basados en instancias y del lazy learning. Su funcionamiento es sencillo, almacena en memoria los datos de entrenamiento y cuando debe hacer una predicción calcula la distancia entre los ejemplos de entrenamiento y la instancia a clasificar y considera los K más cercanos, a partir de ahí se puede decidir la clase de la instancia en función de las clases de los más cercanos por voto mayoritario, aunque esta última predicción podría ponderarse por la distancia de cada ejemplo o por otras técnicas.

7.1.2. Support Vector Machine

Las máquinas de soporte vectorial SVM son clasificadores lineales que tratan de ser óptimos en el sentido de trasladarse a un espacio de alta dimensionalidad (posiblemente infinita) en el que encontrar un hiperplano canónico que maximice el margen de separación del conjunto de datos de entrenamiento, de esta forma trata de ganar generalidad. Hacer este traslado a dicho espacio puede ser costoso en tiempo y espacio, por ello se suelen utilizar funciones kernel (producto interno del espacio) y ahorrarnos especificar la función de transformación al espacio en el que separar. Para obtener el hiperplano y garantizar su generalidad se utiliza teoría de aprendizaje estadístico y optimización de Lagrange.

7.1.3. Regresión Logística

Este método es un modelo lineal que se utiliza para clasificación y obtiene como salida un número entre 0 y 1 que puede entenderse como probabilidad de que un dato pertenezca a una clase, se establece un umbral normalmente en 0.5 a partir del cúal se determina la clase del dato. Se basa en el uso de una función sigmoidal, puede ser basada en la exponencial o incluso hiperbĺica, que suaviza la transición entre las 2 clases, la cual es muy abrupta en la función signo de otros modelos. Al ser un método basado en regresión, trataremos de minimizar una función de error denominada error de entropía cruzada con la siguiente expresión obtenida a través del criterio de máxima verosimilitud:

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} ln(1 + e^{-y_i w^T x_i})$$

Para ello calculamos su gradiente y lo usaremos en el algoritmo de Gradiente Descendente Estocástico.

7.1.4. Naive Bayes

Se trata de un clasificador probabilístico simple que hace una fuerte y casi siempre falsa suposición, la independencia entre las variables. A pesar de ello logra resultados razonables y es un clasificador que se suele usar como punto de partida para realizar comparaciones con otros modelos más complejos. Se utiliza para ello el teorema de Bayes y se calcula la predicción utilizando probabilidades condicionadas a los eventos recogidos en las variables predictoras.

7.1.5. Árbol de decisión

Un árbol de decisión es un clasificador que en función de un conjunto de atributos trata de determinar a que clase pertenece cada instancia. Pueden construirse de muchas formas pero una bastante estándar es utilizar el índice de Gini, una medida de la pureza de un nodo. Se usa para dividir cada nodo aquel atributo que genere un índice Gini más pequeño para poder separar mejor las clases. Este tipo de modelos, aunque simples, tienen buenos resultados pero tienen problemas para manejar valores perdidos y los atributos continuos no se separan de la forma más deseable. Además pueden tender al sobreajuste a entrenamiento si no tenemos suficientes y variados datos. Por otro lado, son combinados en los llamados ensembles más comunes, modelos que agregan clasificadores y son de lo más utilizado hoy en día, a continuación trataremos 2 ejemplos claves de este paradigma.

7.1.6. Random Forest

Random Forest es un algoritmo que representa la técnica bagging de ensemble learning. La idea es construir un conjunto de clasificadores débiles, que suelen ser tomados como aquellos inestables en el sentido de que varían bastante dependiendo del conjunto de datos con los que entrenen. Estos clasificadores se entrenan haciendo un muestreo con reposición de los datos de entrenamiento originales y entrenando cada uno con las muestras resultantes, además se utiliza un número de características aleatorias y reducidas para construir cada árbol débil, finalmente la agregación se realiza prediciendo la clase predicha con mayor frecuencia por los clasificadores débiles. Son interesantes para nuestro problema students porque estan indicados especialmente para datasets de alto número de características.

7.1.7. Boosting

La técnica de boosting se basa en el entrenamiento de una secuencia de clasificadores débiles en el que en cada iteración los algoritmos van mejorando las debilidades del anterior. Vamos a utilizar 2 variedades, uno adaptativo Adaboost y otro basado en gradientes. El algoritmo Adaboost, en cada iteración va identificado los puntos mal clasificados e incrementando sus pesos para que el siguiente clasificador tenga más cuidado con ellos para clasificarlos bien. Finalmente se hace un voto ponderado para el clasificador final en el que se utiliza un coeficiente de confianza para dirimir la importancia. Por otro lado, Gradient Boosting trata de minimizar una función de error mediante aproximación por gradiente descendente a la hora de construir el siguiente clasificador, para ello la función ha de ser diferenciable y en los problemas de clasificación la base matemática esalgo complicada. Esta optimización por gradiente resulta en un clasificador que controla bien el sobreajuste luego cuantas más iteraciones usemps, solemos encontrar mejores soluciones.

7.1.8. Multilayer Perceptron

Finalmente el perceptron multicapa es una red neuronal totalmente conectada bastante básica. La idea es tener un grafo acíclico dirigido de perceptrones con una función de activación diferenciable en lugar de la función signo. La entrada de cada perceptron debe aprender unos pesos por cada variable de entrada para llevar a cabo la decisión. A la hora de entrenar se inicializan los pesos pequeños y aleatorios y se usa el algoritmo de propagación hacia atrás para transmitir el gradiente hasta las capas iniciales e ir ajustando los pesos, se suelen usar grupos de datos llamados minibatches aleatorios para evitar caer en óptimos locales y mejorar la optimización tomando en cada punto una representación más clara de la superficie a optimizar. Esto genera clasificadores lentos de entrenar dependiendo de la arquitectura pero en general bastante eficaces, sobre todo con datos numéricos, aunque tienden a sobreajustar los datos por lo que debemos usar técnicas de regularización.

8. Bibliografía

- Documentación y foro de Knime
- Web de Jorge Casillas. Vídeos de Knime
- https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/
- https://sci2s.ugr.es/imbalanced
- https://medium.com/analytics-vidhya/principal-component-analysis-pca-8a0fcba2e30c