Inteligencia de Negocio (2017-2018)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica 1

Análisis de Ingresos Basado en Datos Socioeconómicos

Javier Béjar Méndez

November 5, 2017

Contents

List of Figures

List of Tables

1 Introducción

Estudiaremos el comportamiento de una serie de algoritmos de aprendizaje supervisado sobre el conjunto de datos Adult para estudiar la influencia de determinados factores demográficos y socioeconómicos sobre la capacidad de ingresos anual de una persona (concretamente, si podrá o no ganar más de 50.000 dólares anuales). El estudio se realizara íntegra y exclusivamente en el entorno Knime[14].

Primero analizaremos los algoritmos y sus resultados obtenidos de manera individual y los efectos de la parametrización correcta en dichos algoritmos, seguidamente haremos una comparación entre los resultados de los distintos algoritmos. Además estudiaremos la importancia del preprocesamiento de los datos y abordaremos la tipología de los mismos.

1.1 Metodología para la validación

A la hora de validar los resultados usaremos la validación cruzada[7], creando 5 conjuntos de datos disjuntos, sobre los que entrenar y validar todos los algoritmos. El workflow de Knime que lo implementa es el siguiente:

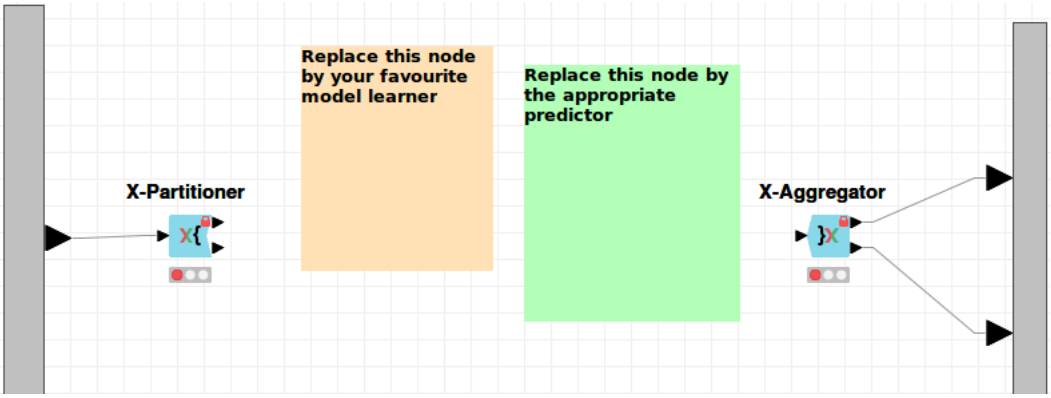

Figure 1.1: Workflow de Knime para la validación cruzada.

Donde añadiremos los correspondientes *learners* y *predictors*.

1.2 Conjunto de datos

Emplearemos el conocido conjunto de datos Adult[2]. Los datos son casos reales extraídos del censo de EE.UU. El conjunto de datos contiene 14 características[3] (6 numéricas y 8 categóricas) y 48.842 ejemplos (algunos de los cuales presenta valores desconocidos) e incluye información sobre la edad, educación, ocupación, sexo, etnia, etc.

1.3 Algoritmos

Se han seleccionado los siguientes algoritmos:

- **Ada Boost**[5] es un algoritmo que combina la salida de otros algoritmos de aprendizaje débil, *weak learning*[12], mediante una suma ponderada. Es sensible al *ruido* y a valores atípicos. En su favor tiene una gran adaptatividad ya que algunos de los algoritmos que combina puede ser mejor en distintos tipos de problemas, creando en su conjunto un modelo rápido con resultados decentes. Se ha utilizado la variante Real AdaBoost, de la librería de Weka[11].

- **C4.5**[6], este algoritmo genera un árbol de decisión mediante el concepto de entropía de información. El algoritmo considera todas las decisiones posibles que pueden dividir el conjunto de datos y selecciona la decisión que aporte mayor ganancia de información para construir el árbol. Permite el manejo de datos con valores desconocidos, ignorandolos, permite trabajar con valores continuos y se basa en el criterio de proporción de ganancia evitando que las variables con mayor número de categorías se beneficien.

- **K-NN**[13], algoritmo de clasificación por los k vecinos más cercanos, que clasifica un dato según la clase mayoritaria de los k vecinos más próximos. Es un algoritmo simple y rápido, no se ajusta bien a fronteras de división complejas y es muy sensible a la naturaleza de los datos, el correcto ajuste de los datos o establecimiento de pesos en los atributos permite obtener muy buenos resultados, por lo que nos servirá para observar la mejora que supone el correcto preprocesamiento de los datos.

- **Random Forest**[15], es un algoritmo que genera una gran cantidad de arboles de decisión y clasifica el dato según la aclasificación que aparezca en el mayor número de arboles. Tiene una excelente capacidad de evitar el sobre-entrenamiento, es muy eficiente en grandes bases de datos con gran cantidad de atributos y aporta información sobre la topología de los datos que puede servir de ayuda a la hora de preprocesar la base de datos, como la importancia de los distintos atributos o el error de generalización.

- **Gradient Boosting**[9], es un algoritmo que produce un modelo de predicción según un conjunto de modelos de predicción débiles. La construcción del modelo se realiza de manera escalonada, es decir, mejora el modelo minimizando el error de dicha función. Se comporta bien en espacios de alta dimensión ya que no es susceptible a las dependencias e interacciones lineales entre los atributos. Requiere una correcta parametrización y presenta cierta tendencia al sobreentrenamiento. Hemos elegido la versión basada en árboles de decisión.

- **MLP**[4], Multi-layer Perceptron, es un modelo de red neuronal que permite definir fronteras de datos no lineares, tiene una capacidad limitada a la hora de evitar mínimos locales y requiere una correcta parametrización para obtener buenos resultados. Se ha elegido este algoritmo para diversificar el tipo de algoritmos elegidos y observar su mejora con una correcta parametrización, más significativa que la aportada por el preprocesamiento de datos, ya que se trata de un algoritmo diseñado para problemas complejos.

Se ha intentado seleccionar algoritmos de tipología diversa, que permitan comprobar distintos aspectos de los datos y la importancia que representa el par naturaleza de los datos y el algoritmo selecionado, asi como la parametrización de los algoritmos y el efecto de preprocesar los datos de entrenamiento.

1.4 Workflow global

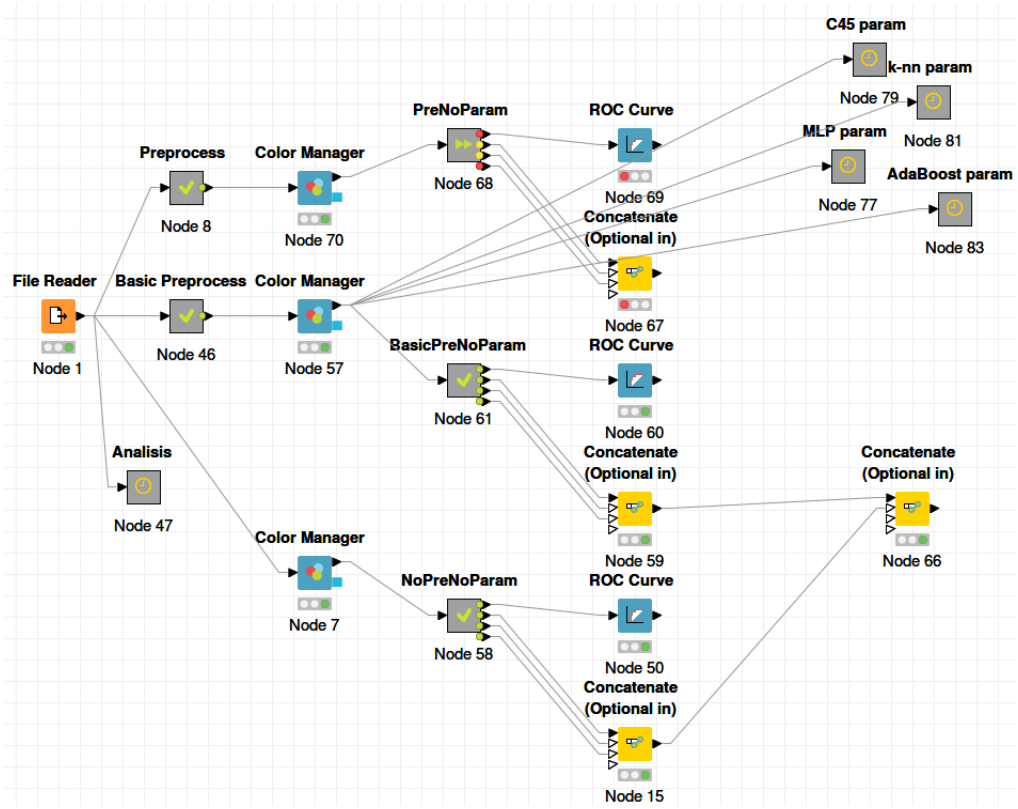

Figure 1.2: Workflow de Knime global.

En esta figura observamos los meta nodos para cada conjunto de algoritmos, los metanodos de preprocesamiento y el metanodo de analisis, que abordaremos en el apartado 6.

El metanodo que engloba todos los algoritmos sigue el siguiente workflow:

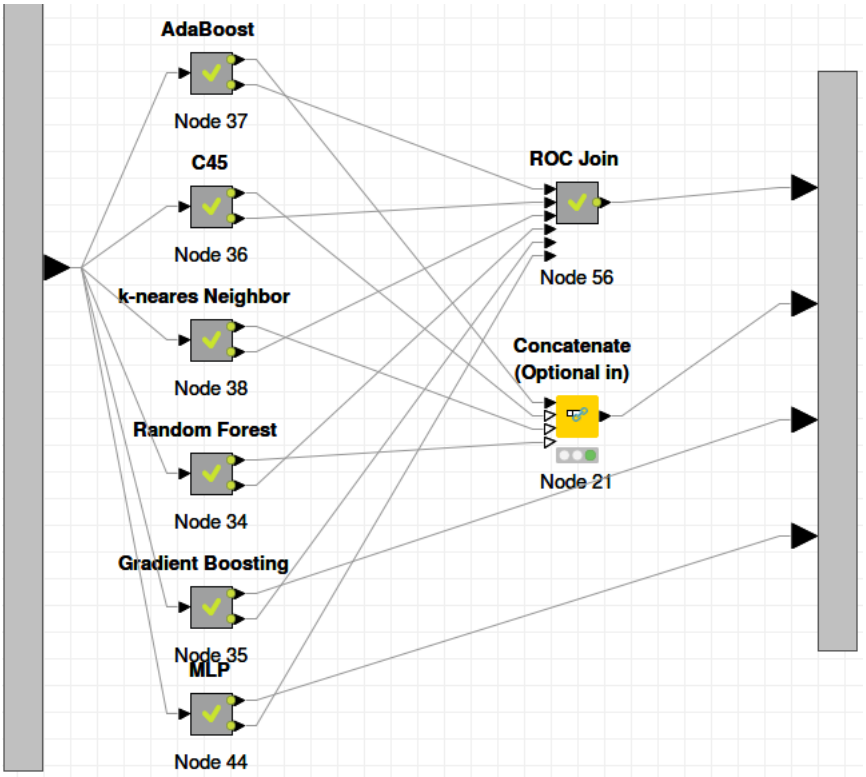

Figure 1.3: Workflow de Knime del metanodo de algoritmos.

El metanodo ROC joins reune los datos para poder obtener la curva ROC de todos los algoritmos superpuestos. Mientras que el concatenate contatena las tablas para poder visualizar los datos estadísticos de todos los algoritmos.

El workflow de cada algoritmo sigue la siguiente estructura:

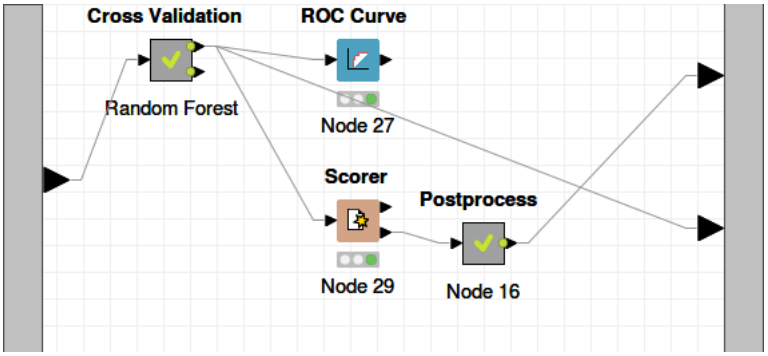

Figure 1.4: Workflow de Knime para los distintos algoritmos.

Que nos permite obtener la curva ROC[1], y mediante Scorer obtenemos la matriz de

confusión y datos estadísticos, el metanodo Postprocess formatea dichas tablas para la visualización global de todos los algoritmos.

2 Resultados Obtenidos

2.1 Ada Boost

El Workflow de Knime es el siguiente:

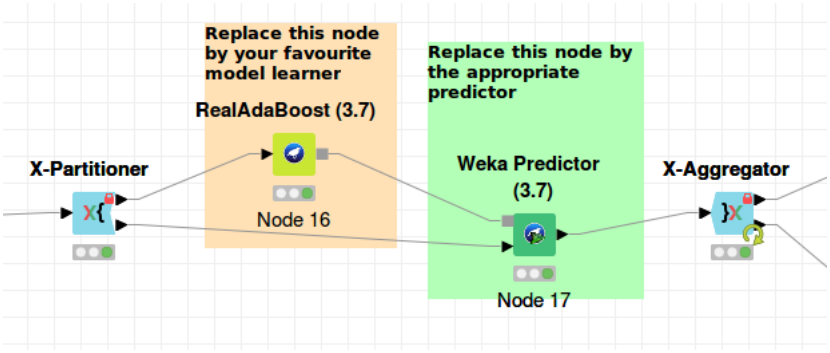

Figure 2.1: Workflow de Knime para la validación cruzada mediante el algoritmo AdaBoost.

La configuración establecida es la que viene por defecto:

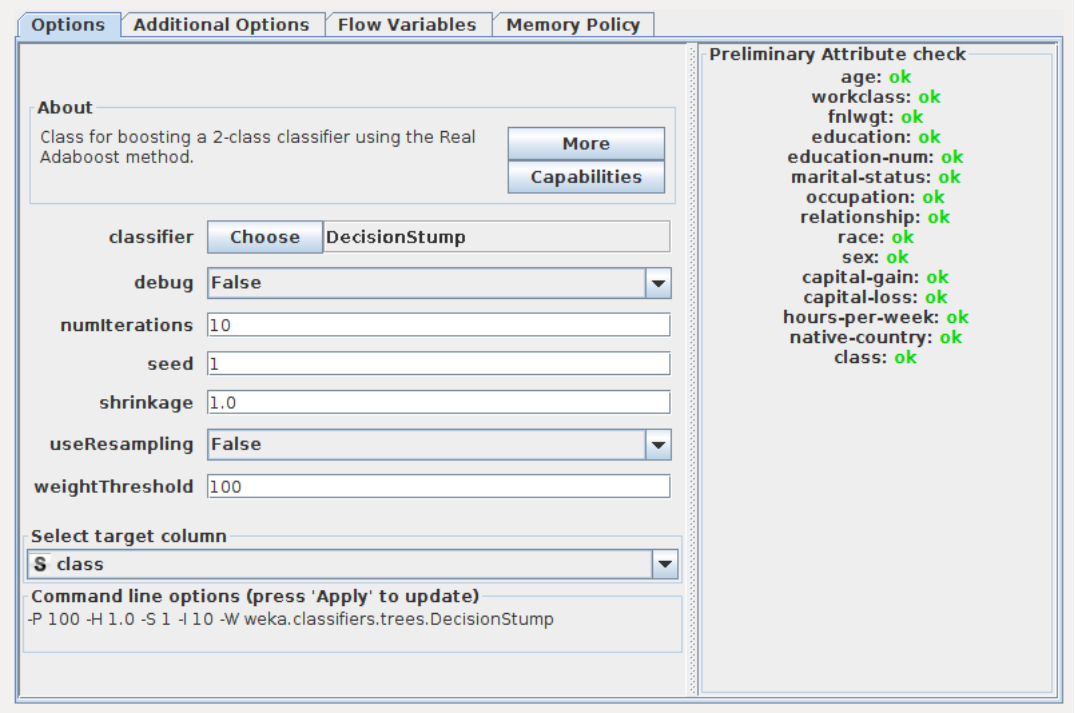

Figure 2.2: Configuración por defecto del algoritmo AdaBoost.

Los resultados obtenidos los observamos en las siguientes figuras:

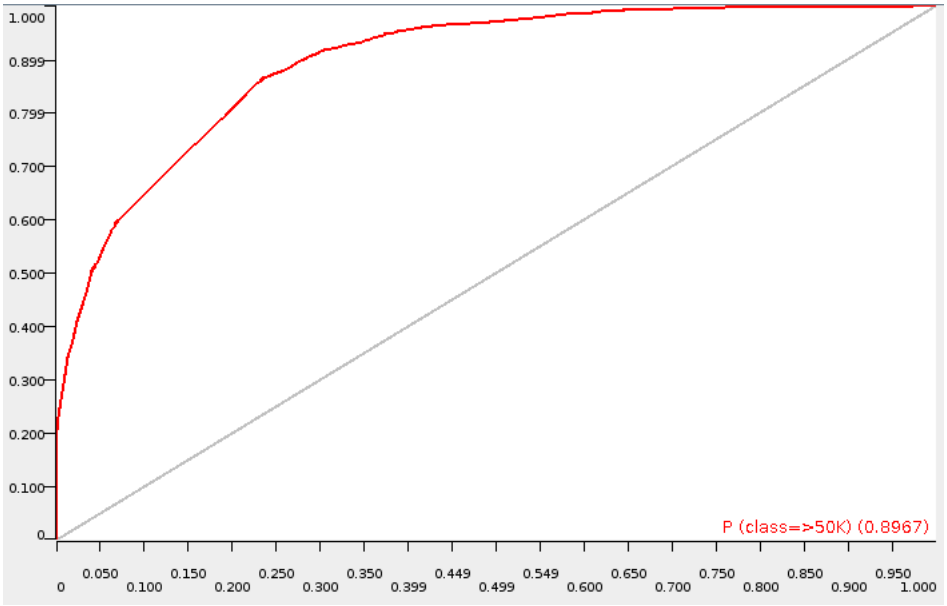

Figure 2.3: Curva ROC de la clasificación obtenida mediante AdaBoost.

Row ID	<=50K	>50K
<=50K	35458	1697
>50K	5635	6052

Figure 2.4: Matriz de confusión con los resultados de AdaBoost.

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specifity	F-measure	Accuracy	Cohen's kappa
<=50K	35458	5635	6052	1697	0.954	0.863	0.954	0.518	0.906	?	?
>50K	6052	1697	35458	5635	0.518	0.781	0.518	0.954	0.623	?	?
Overall	?	?	?	?	?	?	?	?	?	0.85	0.534

Figure 2.5: Datos estadísticos obtenidos por AdaBoost.

2.2 C4.5

El Workflow de Knime es el siguiente:

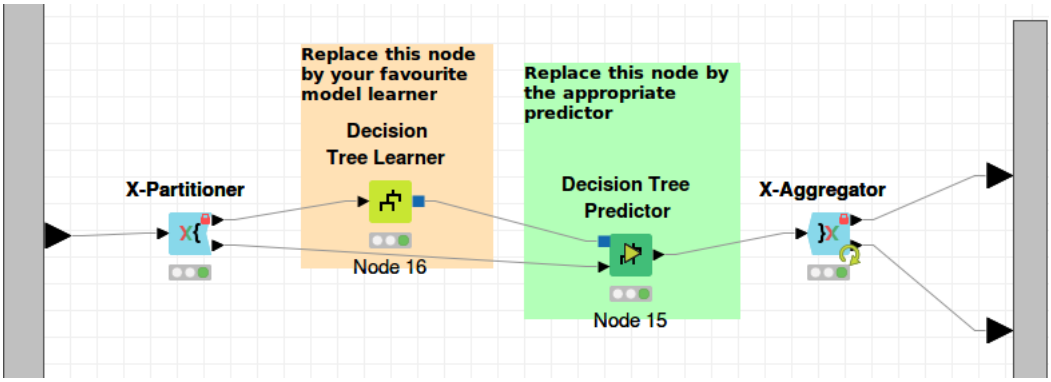

Figure 2.6: Workflow de Knime para la validación cruzada mediante el algoritmo C4.5.

La configuración establecida es la que viene por defecto:

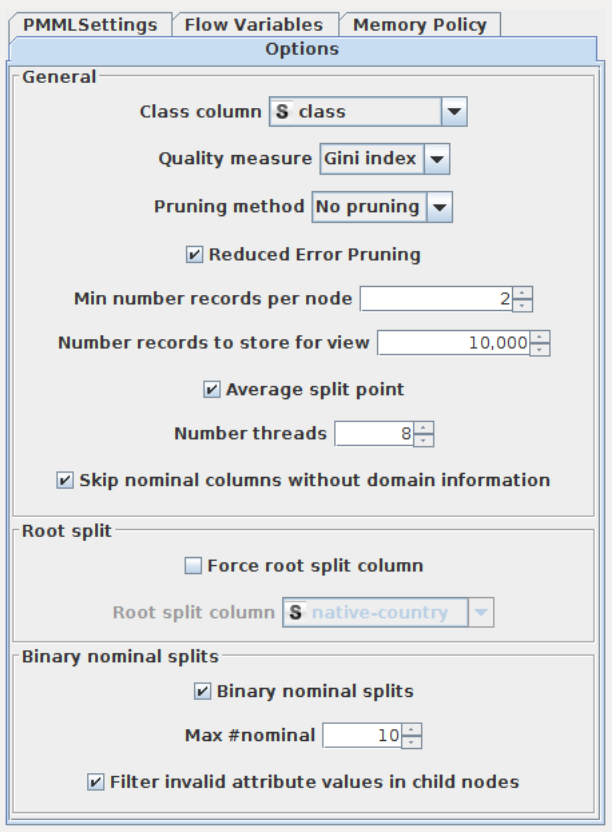

Figure 2.7: Configuración por defecto del algoritmo C4.5.

Los resultados obtenidos los observamos en las siguientes figuras:

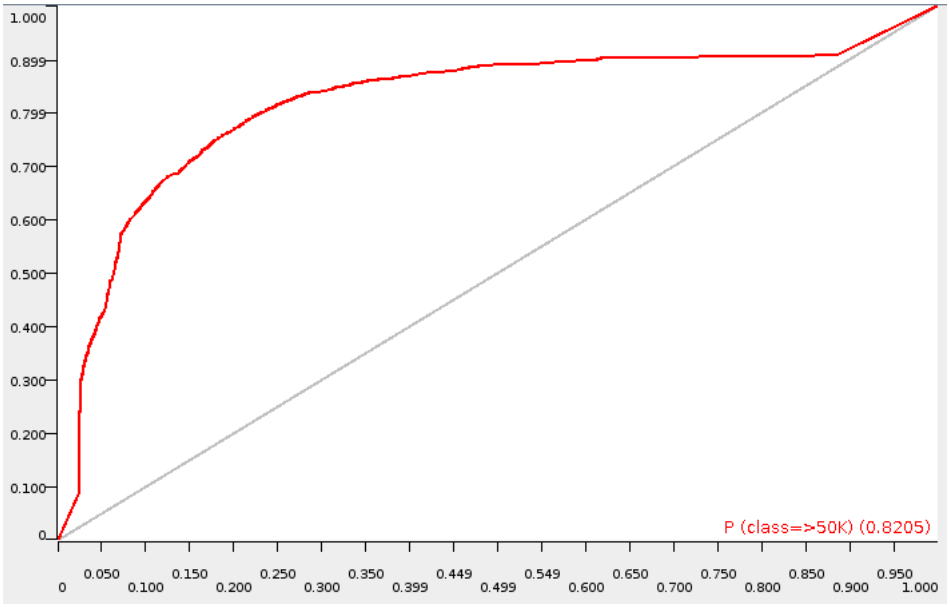

Figure 2.8: Curva ROC de la clasificación obtenida mediante C4.5.

class \ Pr...	<=50K	>50K
<=50K	33363	3792
>50K	4235	7452

Figure 2.9: Matriz de confusión con los resultados de C4.5.

Row ID	I TruePositives	I FalsePositives	I TrueNegatives	I FalseNegatives	D Recall	D Precision	D Sensitivity	D Specifity	D F-measure	D Accuracy	D Cohen's kappa
<=50K	33363	4235	7452	3792	0.898	0.887	0.898	0.638	0.893	?	?
>50K	7452	3792	33363	4235	0.638	0.663	0.638	0.898	0.65	?	?
Overall	?	?	?	?	?	?	?	?	?	0.836	0.543

Figure 2.10: Datos estadísticos obtenidos por C4.5.

2.3 K-NN

El Workflow de Knime es el siguiente:

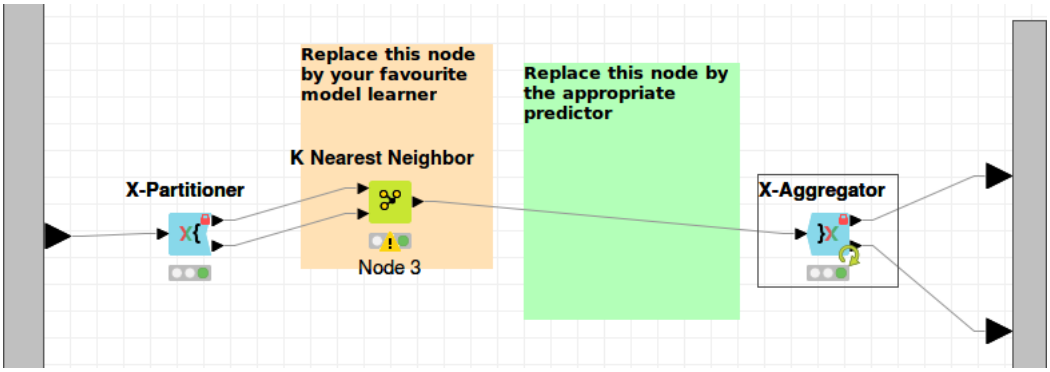

Figure 2.11: Workflow de Knime para la validación cruzada mediante el algoritmo K-NN.

La configuración establecida es la que viene por defecto:

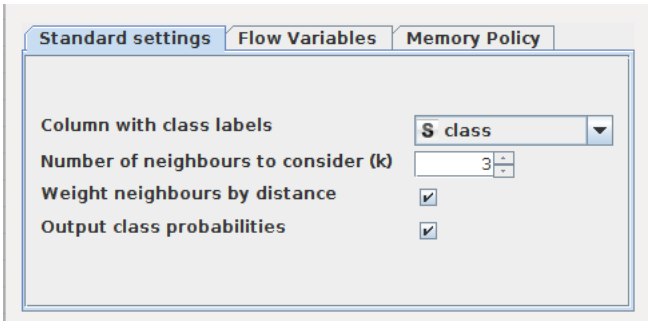

Figure 2.12: Configuración por defecto del algoritmo K-NN.

Los resultados obtenidos los observamos en las siguientes figuras:

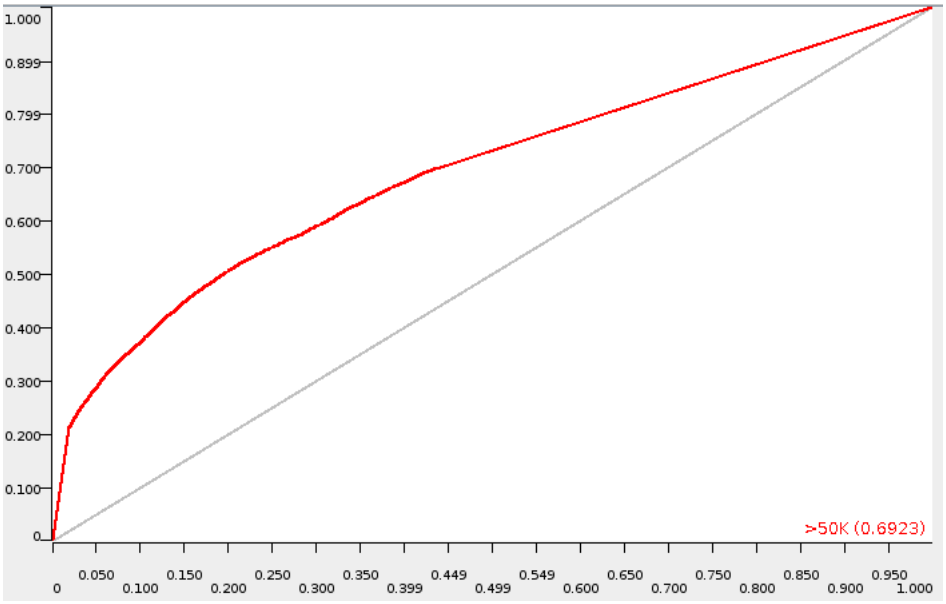

Figure 2.13: Curva ROC de la clasificación obtenida mediante K-NN.

class \ Cl...	<=50K	>50K
<=50K	32046	5109
>50K	6677	5010

Figure 2.14: Matriz de confusión con los resultados de K-NN.

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specifity	F-measure	Accuracy	Cohen's kappa
<=50K	32046	6677	5010	5109	0.862	0.828	0.862	0.429	0.845	?	?
>50K	5010	5109	32046	6677	0.429	0.495	0.429	0.862	0.46	?	?
Overall	?	?	?	?	?	?	?	?	?	0.759	0.305

Figure 2.15: Datos estadísticos obtenidos por K-NN.

2.4 Random Forest

El Workflow de Knime es el siguiente:

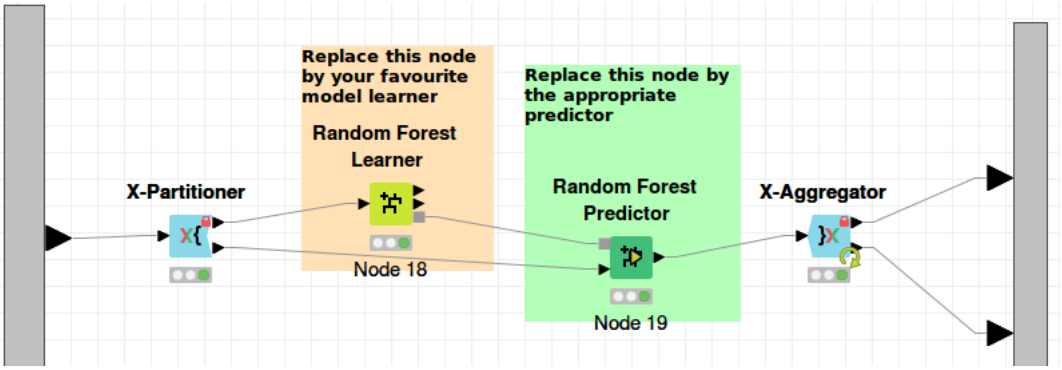

Figure 2.16: Workflow de Knime para la validación cruzada mediante el algoritmo Random Forest.

La configuración establecida es la que viene por defecto:

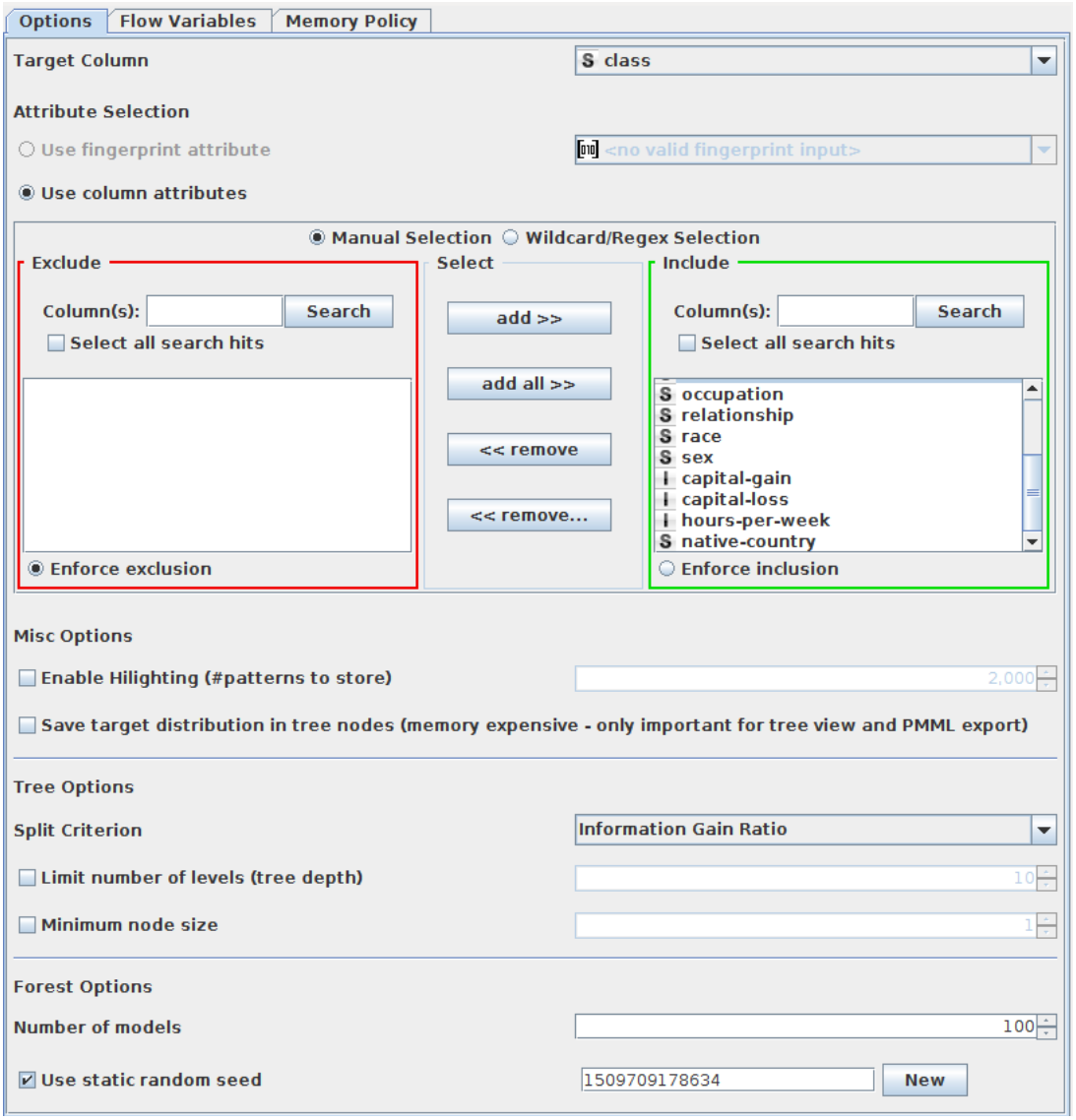

Figure 2.17: Configuración por defecto del algoritmo Random Forest.

Los resultados obtenidos los observamos en las siguientes figuras:

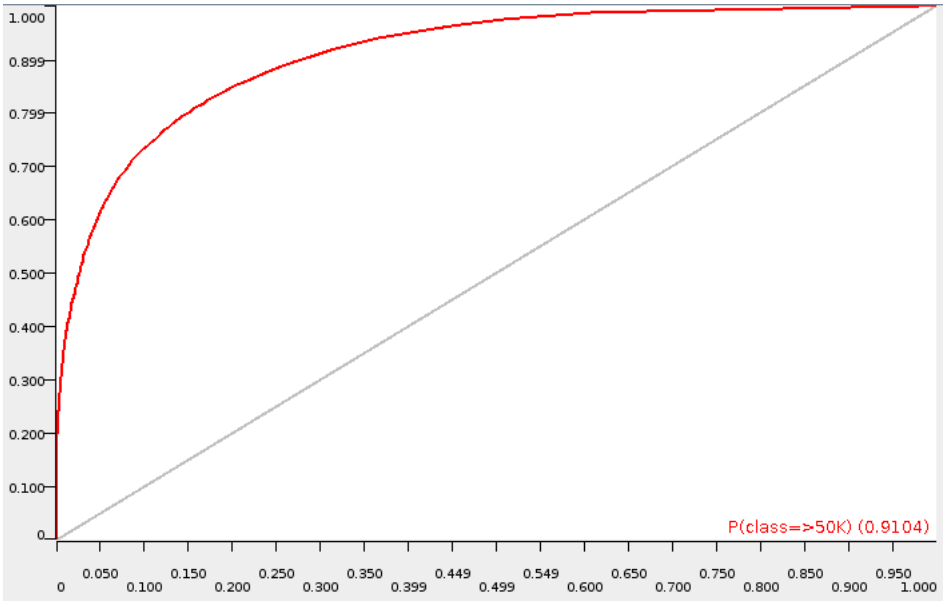

Figure 2.18: Curva ROC de la clasificación obtenida mediante Random Forest.

Row ID	<=50K	>50K
<=50K	35322	1833
>50K	4548	7139

Figure 2.19: Matriz de confusión con los resultados de Random Forest.

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specifity	F-measure	Accuracy	Cohen's kappa
<=50K	35322	4548	7139	1833	0.951	0.886	0.951	0.611	0.917	?	?
>50K	7139	1833	35322	4548	0.611	0.796	0.611	0.951	0.691	?	?
Overall	?	?	?	?	?	?	?	?	?	0.869	0.61

Figure 2.20: Datos estadísticos obtenidos por Random Forest.

2.5 Gradient Boosting

El Workflow de Knime es el siguiente:

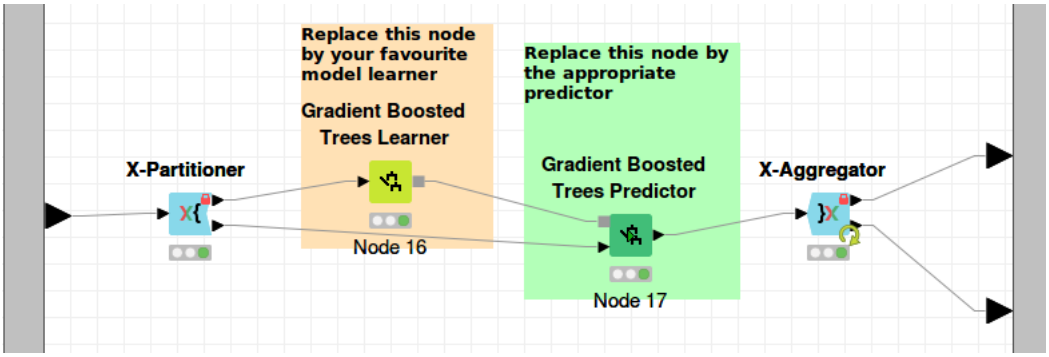

Figure 2.21: Workflow de Knime para la validación cruzada mediante el algoritmo Gradient Boosting.

La configuración establecida es la que viene por defecto:

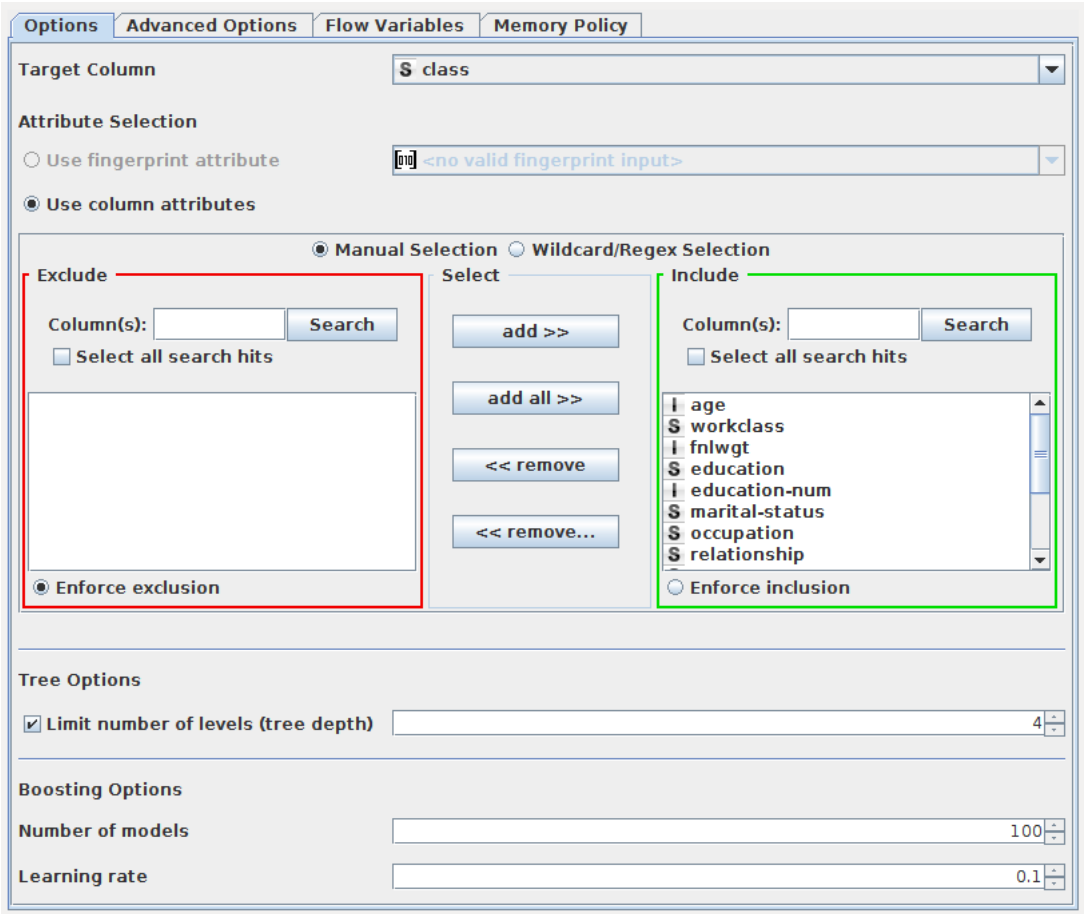

Figure 2.22: Configuración por defecto del algoritmo Gradient Boosting.

Los resultados obtenidos los observamos en las siguientes figuras:

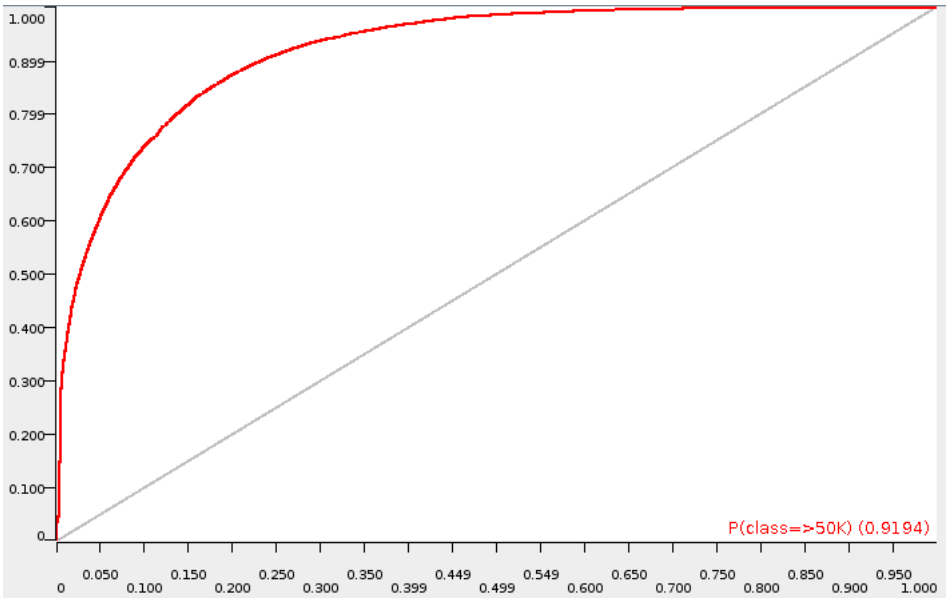

Figure 2.23: Curva ROC de la clasificación obtenida mediante Gradient Boosting.

Row ID	<=50K	>50K
<=50K	35036	2119
>50K	4345	7342

Figure 2.24: Matriz de confusión con los resultados de Gradient Boosting.

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specifity	F-measure	Accuracy	Cohen's kappa
<=50K	35036	4345	7342	2119	0.943	0.89	0.943	0.628	0.916	?	?
>50K	7342	2119	35036	4345	0.628	0.776	0.628	0.943	0.694	?	?
Overall	?	?	?	?	?	?	?	?	?	0.868	0.611

Figure 2.25: Datos estadísticos obtenidos por Gradient Boosting.

2.6 MLP

Para este algoritmo es necesario un preproceso de datos para eliminar todas las variables cualitativas (mediante el módulo one to many), ya que solo acepta variables numericas (double).

El Workflow de Knime es el siguiente:

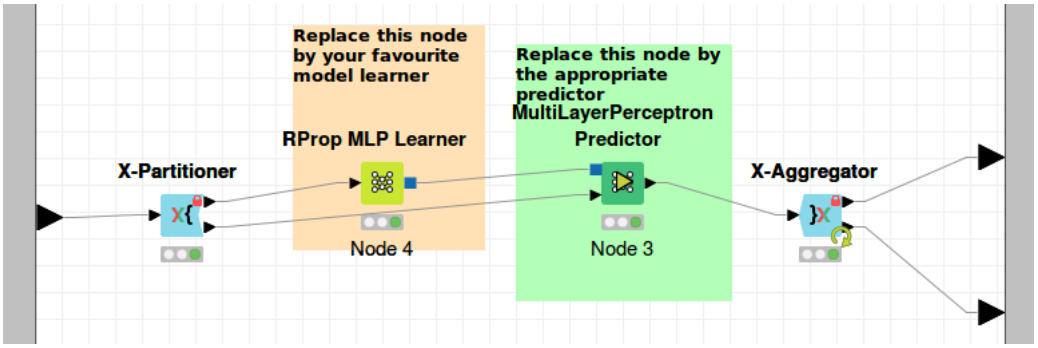

Figure 2.26: Workflow de Knime para la validación cruzada mediante el algoritmo MLP.

La configuración establecida es la que viene por defecto:

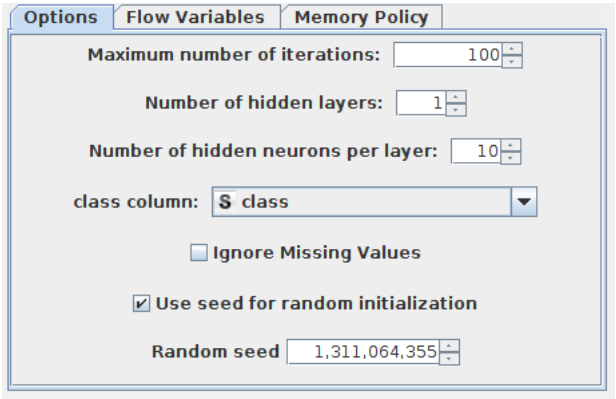

Figure 2.27: Configuración por defecto del algoritmo MLP.

Los resultados obtenidos los observamos en las siguientes figuras:

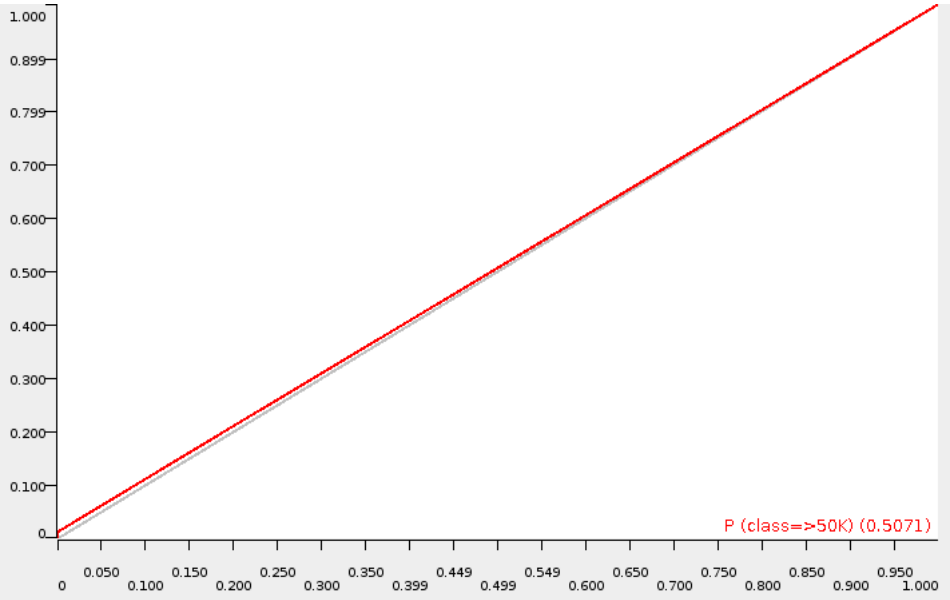

Figure 2.28: Curva ROC de la clasificación obtenida mediante MLP.

class \ Pr...	<=50K	>50K
<=50K	11649	1
>50K	11515	172

Figure 2.29: Matriz de confusión con los resultados de MLP.

Row ID	I TruePositives	I FalsePositives	I TrueNegatives	I FalseNegatives	D Recall	D Precision	D Sensitivity	D Specifity	D F-measure	D Accuracy	D Cohen's kappa
<=50K	11649	11515	172	1	1	0.503	1	0.015	0.669	?	?
>50K	172	1	11649	11515	0.015	0.994	0.015	1	0.029	?	?
Overall	?	?	?	?	?	?	?	?	?	0.507	0.015

Figure 2.30: Datos estadísticos obtenidos por MLP.

3 Análisis de Resultados

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specifity	F-measure	Accuracy
AdaBoost	6052	1697	35458	5635	0.518	0.781	0.518	0.954	0.623	0.85
C4.5	7452	3792	33363	4235	0.638	0.663	0.638	0.898	0.65	0.836
K-NN	5010	5109	32046	6677	0.429	0.495	0.429	0.862	0.46	0.759
Random F...	7139	1833	35322	4548	0.611	0.796	0.611	0.951	0.691	0.869
Gradient B...	7342	2119	35036	4345	0.628	0.776	0.628	0.943	0.694	0.868
MLP	172	1	11649	11515	0.015	0.994	0.015	1	0.029	0.507

Figure 3.1: Datos estadísticos obtenidos para todos los algoritmos.

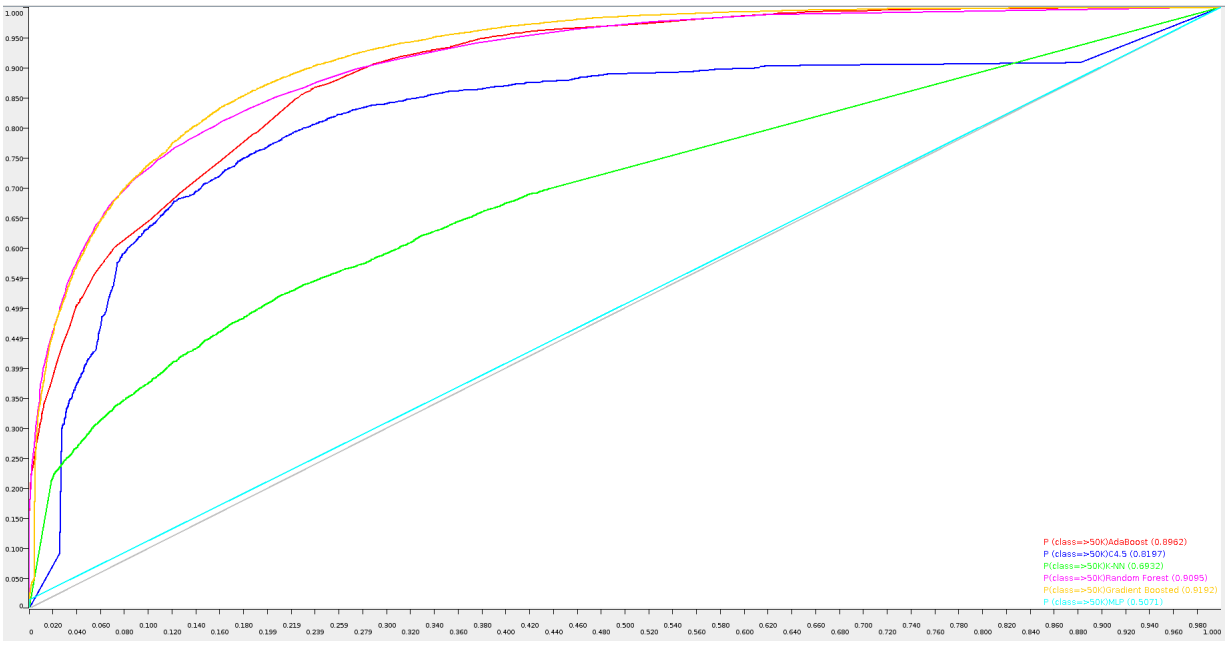

Figure 3.2: Curva ROC comparativa entre todos los algoritmos.

Estos resultados se han realizado con la base de datos original sin modificar, a excepción del MLP ya que solo puede trabajar con valores numéricos, en la que se ha realizado un preprocesado ya descrito anteriormente.

Se ha definido como clase positiva la clase $P(class \geqslant 50)$, siendo esta la que se muestra en las curvas ROC y datos estadísticos.

A la vista de los resultados obtenidos podemos discutir lo siguiente sobre los algoritmos:

- El algoritmo K-NN obtiene una valoración mediocre respecto al resto, con un porcentaje de acierto global del 75.9%(accuracy), esto se debe principalmente a que los atributos no están normalizados, repercutiendo en que los atributos con dominios de valor muy altos respecto a los demás atributos cobren mayor relevancia, y que no es capaz de procesar los atributos nominales, ignorando una parte sustancial de información. Esta falta de información y asignación incorrecta de atributos relevantes hacen que la frontera de datos entre las dos clases este muy poco definida, lo observamos en que obtiene una precisión del 49.5% que nos dice que asigna muchos muchos falsos positivos y un recall de 42.9%, asignando muchos falsos negativos. El algoritmo es más especifico, ya que asigna más falsos negativos $P(class \leqslant 50)$, por lo tanto sería un algoritmo conservador en cuanto a que prefiere discriminar a la población positiva.

- El algoritmo MLP obtiene un porcentaje de acierto del 50.7%. Esto se debe a que, análogamente al K-NN, requiere de datos normalizados para su correcto funcionamiento. En este caso a la hora de ajustar los pesos de la red neuronal los atributos con valores muy altos cobran mayor relevancia, sumado a que éste algoritmo requiere de una correcta parametrización, justifican el nefasto resultado obtenido. Analizaremos este algoritmo en mayor detalle en los apartados de parametrización y preprocesamiento, ya que a la vista de los resultados no podemos concretar nada relevante, salvo que clasifica prácticamente a todas las tuplas como negativos (especificidad del 100% y sensibilidad del 1.5%).

- Tenemos a los algoritmos AdaBoost, Random Forest y Gradient Boost con resultados similares, estos algoritmos pertenecen a la categoria de *ensemble learning*[8], por lo que presentan un comoportamiento adaptable, no viendose demasiado afectados por el desbalanceado de clases que presenta la base de datos, podemos observar esto en que los algoritmos, aunque presenten una muy alta especificidad ($\approx 95\%$), mantienen un valor notable de F-measure, entre el 62.3% y el 69.4%, que nos dice que el ratio de verdaderos y falsos positivos es bueno, es decir no pierde demasiada precisión en la clase minoritaria. Entre estos algoritmos el Random Forest y el Gradient Boosted estan a la par (a falta de tiempo para corroborar esta afirmación mediante un test-t[10]),ya que estos presentan modelos más complejos que el AdaBoost, el cual se basa en algoritmos de aprendizaje débil ya predefinidos, y se ven menos influenciados por el desbalanceado de la base de datos. También podemos ver que el comportamiento de los tres algoritmos es mejor al resto observando las curvas ROC, que están mas equilibradas que el resto de algoritmos y cubren un area mayor, siendo el GradientBoosted el algoritmo con una mayor area (91.92%) y presentando la curva más equilibrada, luego con los datos obtenidos podemos decir que ha sido el algoritmo que mejor se ha adaptado al problema.

- Por último está el algoritmo C4.5, que se adapta notablemente al problema, con un porcentaje de acierto del 83.6%, debido a que el modelo de construcción del arbol lidia bien con las bases desbalanceadas, como los algoritmos del punto anterior, sin embargo no es tan fino como dichos algoritmos, esto lo observamos en que aunque

mantiene un nivel similar de sensibilidad, pierde en especificidad.

La base de datos no es de calidad, debido a que no se ha modificado para ajustarse mejor a los requerimientos de los algoritmos, presenta un desbalanceado hacia la clase negativa, contiene valores nominales y numéricos y no está normalizada. Por lo que podemos concluir que los algoritmos que mejor resultado han obtenido son los más adaptables y que lidian con el desbalanceado de clases, destacando entre estos el RandomForest y el GradientBoosted, ya que presentan modelos más complejos basados en árboles de decisión, que se adaptan muy bien a los actuales problemas de la base de datos.

4 Configuración de Algoritmos

Para todos los algoritmos se usará el preprocesado de datos básico, descrito en el apartado 5.

4.1 C4.5

En este algoritmo vamos a ver si la poda del arbol de decisión mejora los resultados obtenidos, para ello cambiaremos en el tree learner predictor que realice poda mediante el algoritmo MDL, obteniendo los siguientes resultados:

Row ID	TrueP...	FalseP...	TrueN...	False...	Recall	Precisi...	Sensit...	Specifity	F-mea...	Accur...
C4.5	9147	2575	9075	2540	0.783	0.78	0.783	0.779	0.781	0.781
C4.5_I	9842	2355	9295	1845	0.842	0.807	0.842	0.798	0.824	0.82

Figure 4.1: Resultados estadísticos comparativos entre c45 con poda y sin poda.

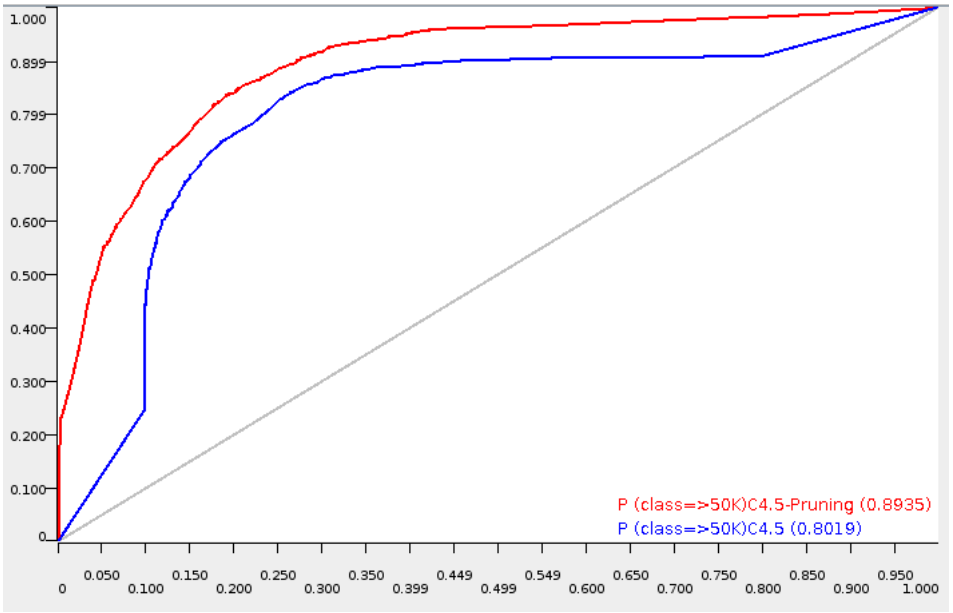

Figure 4.2: Resultados curva ROC comparativos entre c45 con poda y sin poda.

Observamos que con la poda es capaz de refinar el arbol de decisión notablemente (mayor profundidad), ganando un 02.1% de tasa de hacierto global y aumentando notablemente la precisión y sensibilidad, la curva ROC cubre un $\approx 9\%$ más de area.

4.2 K-NN

Vamos a pasar del 3-NN que es el algoritmo actual al 5-NN a ver si al aumentar los vecinos es capaz de detectar mejor las fronteras y así aummentar la tasa de clasificación global y la cobertura de la curva ROC. Los resultados obtenidos son los siguientes:

Row ID	I TrueP...	I FalseP...	I TrueN...	I False...	D Recall	D Precisi...	D Sensit...	D Specifity	D F-mea...	D Accur...
K-NN	9273	2706	8944	2414	0.793	0.774	0.793	0.768	0.784	0.781
K-NN_I	9111	2763	8887	2576	0.78	0.767	0.78	0.763	0.773	0.771

Figure 4.3: Resultados estadísticos comparativos entre 3-NN(K-NN-I en la tabla) y 5-NN(K-NN en la tabla).

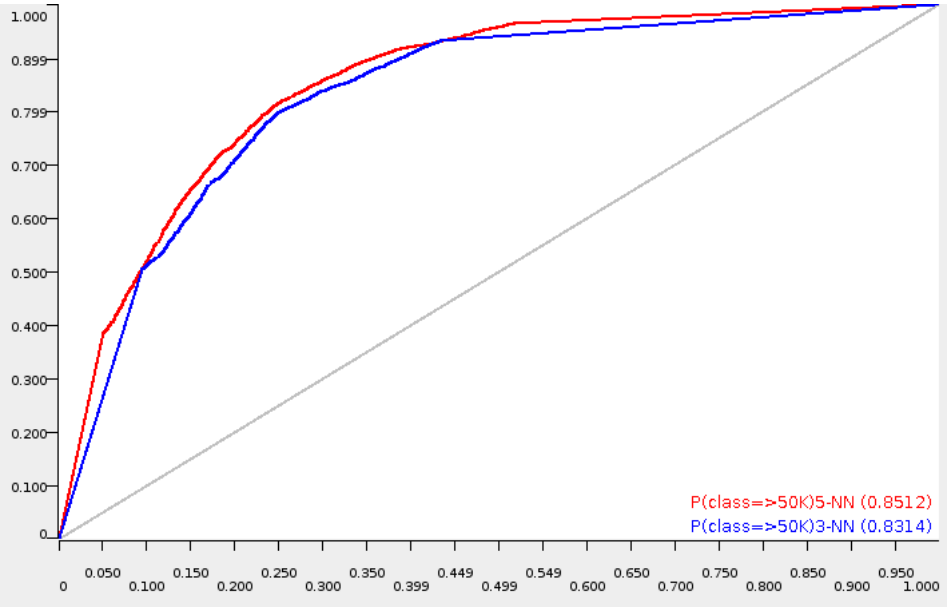

Figure 4.4: Resultados curva ROC comparativos entre 3-NN y 5-NN.

El aumento no es significativo en la tasa de clasificación, y en la curva ROC gana un $\approx 1.9\%$ de cobertura.

4.3 MLP

Nos vamos a centrar en la mejora del algoritmo MPL, se irá modificando un parámetro o una serie de parámetros y se analizarán los resultados obtenidos sucesivamente. Los parámetros que podemos modificar los encontramos en la siguiente figura:

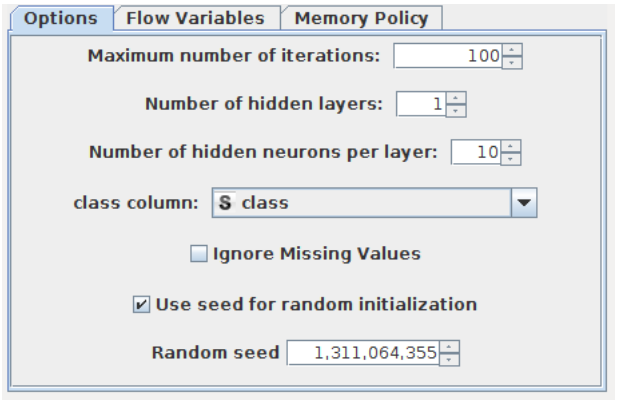

Figure 4.5: Configuración por defecto del algoritmo MLP.

Vamos a modificar el parámetro de capas ocultas, lo ponemos a 2 capas y a 4 capas, obteniendo los siguientes resultados:

Row ID	TrueP...	FalseP...	TrueN...	False...	Recall	Precisi...	Sensit...	Specifity	F-mea...	Accur...
MLP	9941	2439	9211	1746	0.851	0.803	0.851	0.791	0.826	0.821
MLP_I	9993	2472	9178	1694	0.855	0.802	0.855	0.788	0.828	0.821
MLP_I_I	10054	2667	8983	1633	0.86	0.79	0.86	0.771	0.824	0.816

Figure 4.6: Resultados estadísticos comparativos entre MLP, MLP-v1 y MLP-v2.

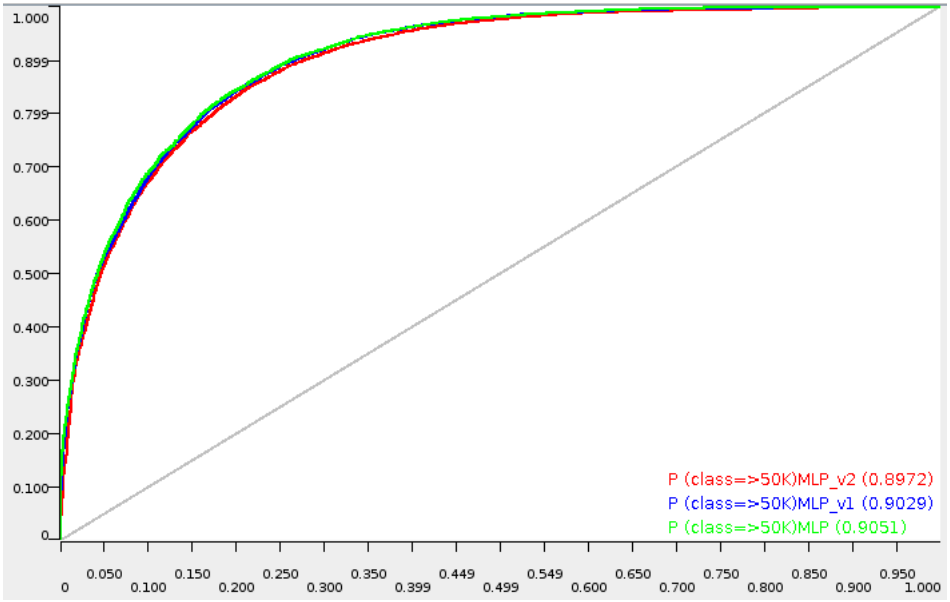

Figure 4.7: Resultados curva ROC comparativos entre MLP, MLP-v1 y MLP-v2.

Observamos que a medida que con 2 capas mejora la el valor F1, sacrificando un poco de especificidad a cambio de mayor sensibilidad, con 4 capas empeora por lo que no seguiremos aumentando el número de capas.

Vamos a probar aumentando el número de neuronas ocultas por capa, a 10 8 y 12 respectivamente para las versiones MLP-v1, MLP-v1.1 y MLP-v1.2, para un valor de capas ocultas igual a 2:

Row ID	TrueP...	FalseP...	TrueN...	False...	Recall	Precisi...	Sensit...	Specifity	F-mea...	Accur...
MLP	9953	2489	9161	1734	0.852	0.8	0.852	0.786	0.825	0.819
MLP_I	9941	2439	9211	1746	0.851	0.803	0.851	0.791	0.826	0.821
MLP_I_I	10018	2444	9206	1669	0.857	0.804	0.857	0.79	0.83	0.824

Figure 4.8: Resultados estadísticos comparativos entre MLP-v1, MLP-v1.1 y MLP-v1.2.

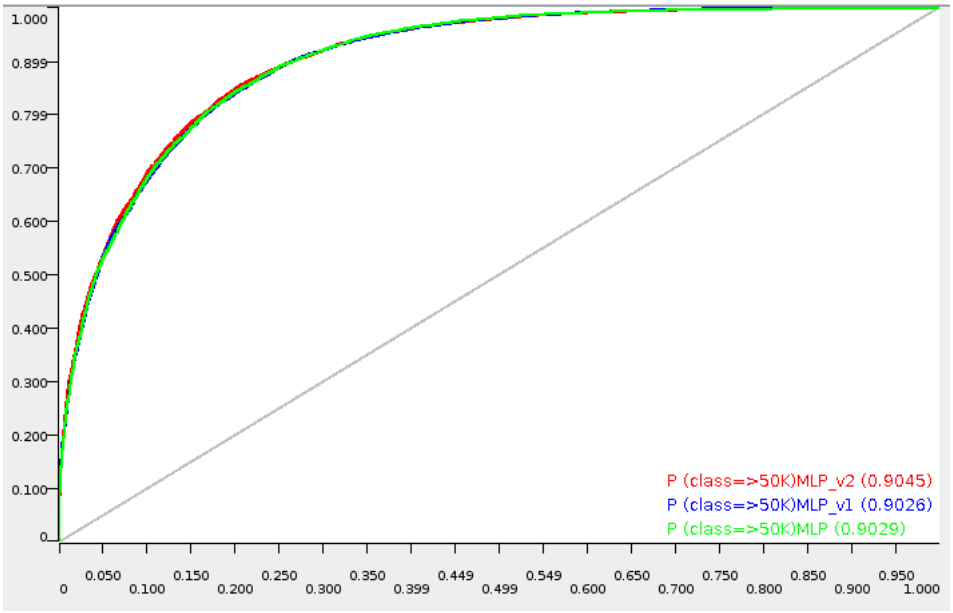

Figure 4.9: Resultados curva ROC comparativos entre MLP-v1, MLP-v1.1 y MLP-v1.2.

Aunque la ultima versión obtiene una pequeña mejoría, son cambios muy pequeños como para saber con certeza si ha mejorado.

5 Procesado de Datos

En este apartado vamos a observar con más detalle los problemas descritos sobre los datos en el apartado 3. Se ha definido el siguiente workflow en Knime para el estudio estadísto:

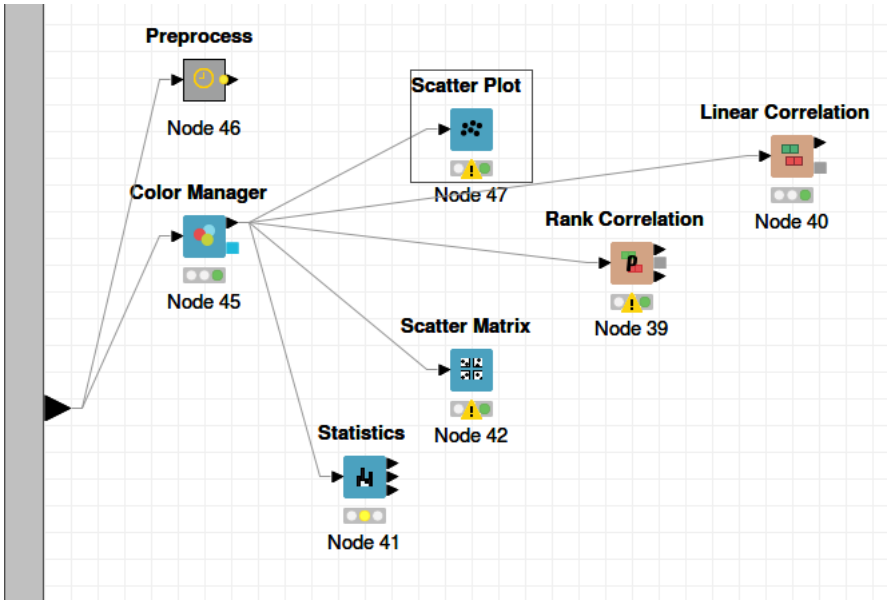

Figure 5.1: Metanodo de analisis de datos en Knime.

Estos nodos nos ofrecen herramientas para obtener información acerca de la base de datos, así, como observamos en los resultados análiticos uno de los problemas es el desbalanceado de clases, observamos esto más detalladamente en la siguiente figura:

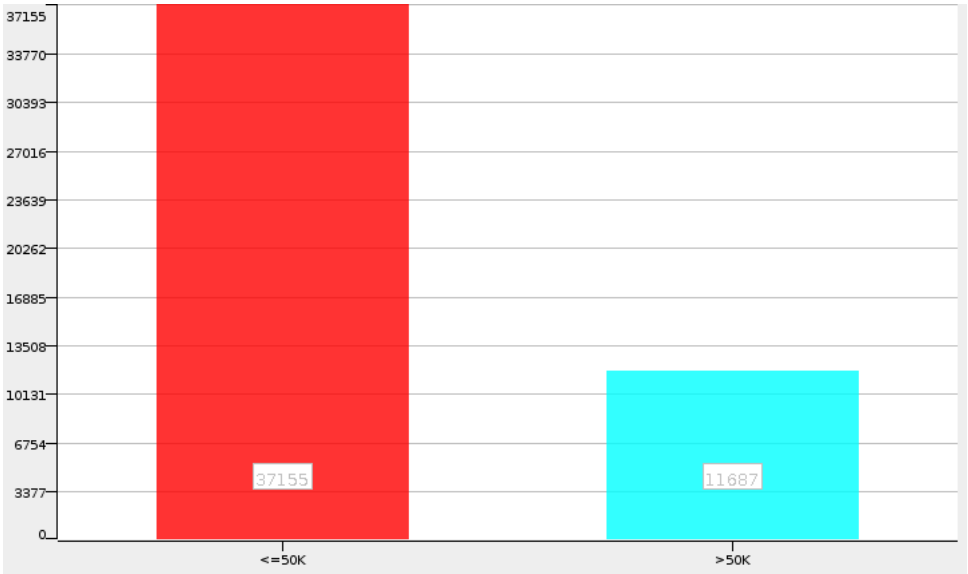

Figure 5.2: Histograma con la distribución del atributo class.

Observamos que hay $37155(\approx 77.06\%)$ de datos etiquetados como $class \leqslant 50$ y $11687(\approx 23,92\%)$ etiquetados como $class \geqslant 50$. Para lidiar con este problema lo haremos me-

diante el módulo *Equal Size Sampling*, que nos dejara la base de datos con la siguiente distribución:

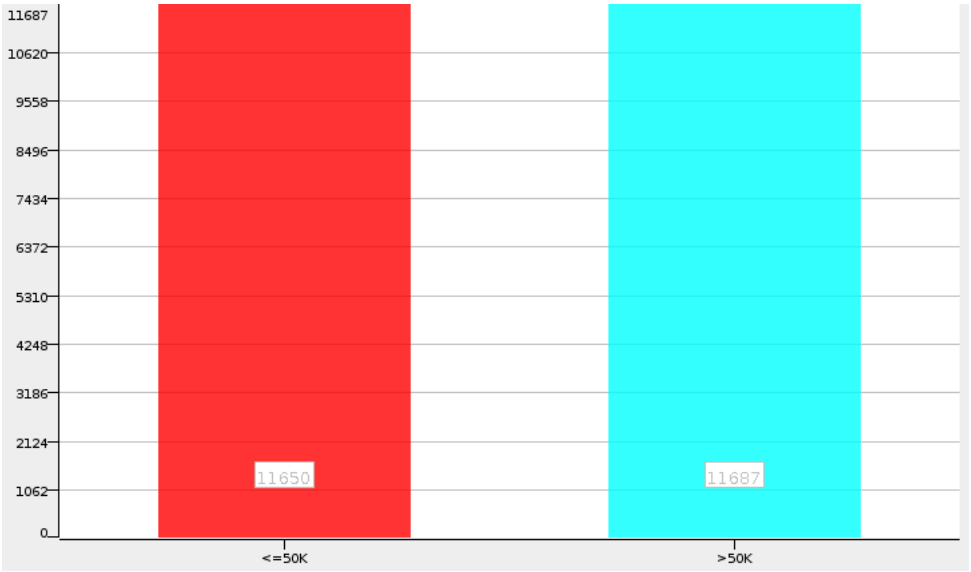

Figure 5.3: Histograma con la distribución del atributo class, balanceada.

Con esto conseguimos una base de datos balanceada a cambio de perder una gran cantidad de datos(25505). En este caso se ha seleccionado aleatoriamente, lo más óptimo sería estudiar que tuplas son las más significativas para incluirlas en la bolsa de datos finales.

También solventaremos el problema de la mezcla de tipos de datos nominales y numéricos, mediante el módulo *One to Many* que añade un atributo nuevo de tipo binario por cada valor presente en los atributos nominales, y la posterior eliminación de los atributos nominales. Y normalizaremos la base de datos según el método min-max, luego el preprocesado de datos básico es el siguiente:

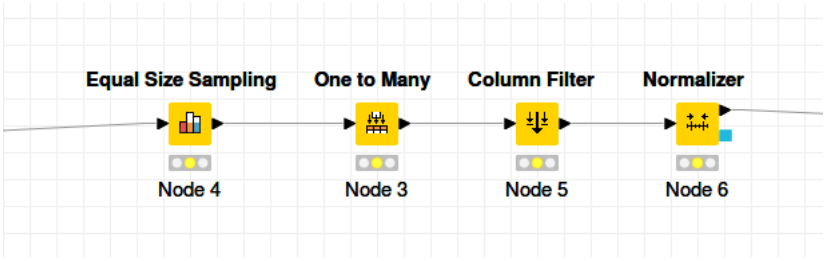

Figure 5.4: Workflow del preprocesado básico de datos.

Ejecutamos todos los algoritmos con este preprocesamiento y con los mismos parámetros, obteniendo los siguientes datos:

Row ID	TruePositives	FalsePositives	TrueNegatives	FalseNegatives	Recall	Precision	Sensitivity	Specifity	F-measure	Accuracy
AdaBoost	10064	2674	8976	1623	0.861	0.79	0.861	0.77	0.824	0.816
C4.5	9147	2575	9075	2540	0.783	0.78	0.783	0.779	0.781	0.781
K-NN	9111	2763	8887	2576	0.78	0.767	0.78	0.763	0.773	0.771
Random F...	10086	2264	9386	1601	0.863	0.817	0.863	0.806	0.839	0.834
Gradient B...	10138	2160	9490	1549	0.867	0.824	0.867	0.815	0.845	0.841
MLP	9993	2472	9178	1694	0.855	0.802	0.855	0.788	0.828	0.821
AdaBoost_...	6052	1697	35458	5635	0.518	0.781	0.518	0.954	0.623	0.85
C4.5_NoPre	7452	3792	33363	4235	0.638	0.663	0.638	0.898	0.65	0.836
K-NN_NoPre	5010	5109	32046	6677	0.429	0.495	0.429	0.862	0.46	0.759
Random F...	7139	1833	35322	4548	0.611	0.796	0.611	0.951	0.691	0.869
Gradient B...	7342	2119	35036	4345	0.628	0.776	0.628	0.943	0.694	0.868
MLP_NoPre	172	1	11649	11515	0.015	0.994	0.015	1	0.029	0.507

Figure 5.5: Datos estadísticos obtenidos para todos los algoritmos con preprocesamiento básico(blanco), comparandolos con los resultados sin preprocesamiento(gris).

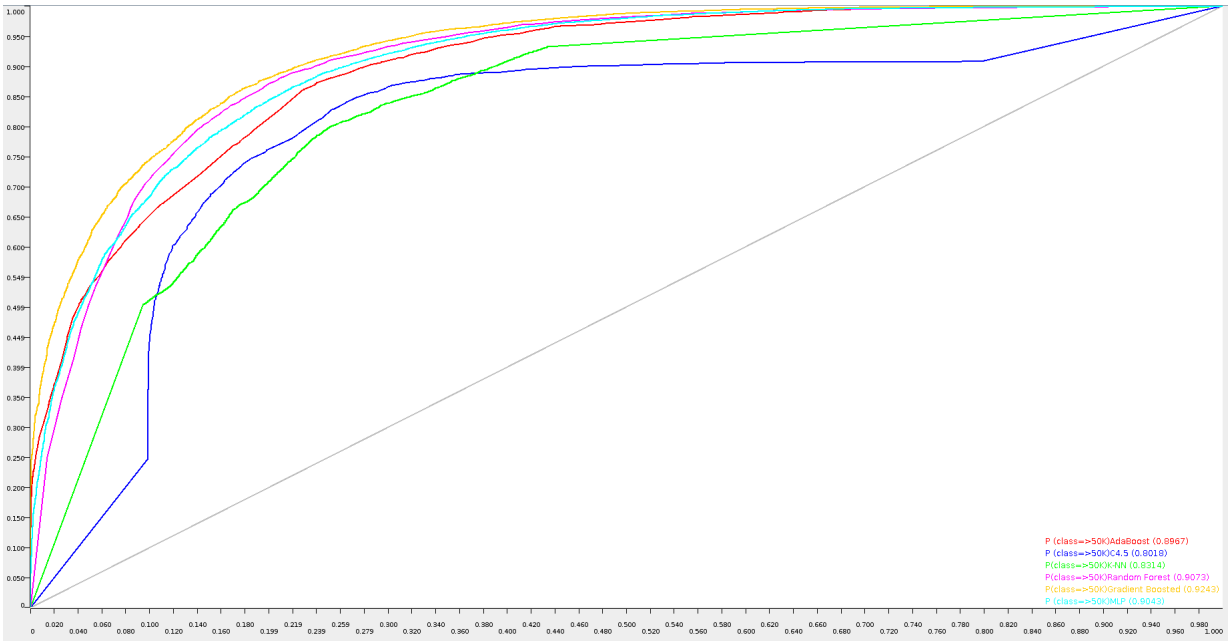

Figure 5.6: Curva ROC comparativa entre todos los algoritmos con preprocesamiento básico.

Las ejecuciones han sido notablemente rápidas, como era de esperar el algoritmo K-NN ha mejorado debido a la normalización he inclusión del conocimiento que aportan las variables nominales en forma de variables binarias, y sobre todo notamos la mejoría en el algoritmo PML, que ahora se situa en los puestos altos de los resultados, los demás algoritmos han obtenido resultados ligeramente peores, debido a la perdida de conocimiento

que ha supuesto el balanceado de los datos. La inflacción del F-measure se debe a que al estar balanceada la base de datos se igualan el recall y la precisión, para asegurarnos si mejoran entrenando con la base de datos balanceada tendriamos que validar con la base completa.

Ahora ejecutaremos el mismo preprocesamiento ha excepción del balanceado, obteniendo los siguientes resultados respecto a las ejecuciones sin preprocesamiento alguno:

Row ID	I TruePositives	I FalsePositives	I TrueNegatives	I FalseNegatives	D Recall	D Precision	D Sensitivity	D Specifity	D F-measure	D Accuracy
AdaBoost	6050	1694	35461	5637	0.518	0.781	0.518	0.954	0.623	0.85
C4.5	7261	4030	33125	4426	0.621	0.643	0.621	0.892	0.632	0.827
K-NN	6571	4169	32986	5116	0.562	0.612	0.562	0.888	0.586	0.81
Random Forest	7161	1914	35241	4526	0.613	0.789	0.613	0.948	0.69	0.868
Gradient Boosting	7351	1948	35207	4336	0.629	0.791	0.629	0.948	0.701	0.871
MLP	9993	2472	9178	1694	0.855	0.802	0.855	0.788	0.828	0.821
AdaBoost_NoPre	6052	1697	35458	5635	0.518	0.781	0.518	0.954	0.623	0.85
C4.5_NoPre	7452	3792	33363	4235	0.638	0.663	0.638	0.898	0.65	0.836
K-NN_NoPre	5010	5109	32046	6677	0.429	0.495	0.429	0.862	0.46	0.759
Random Forest_NoPre	7139	1833	35322	4548	0.611	0.796	0.611	0.951	0.691	0.869
Gradient Boosting_NoPre	7342	2119	35036	4345	0.628	0.776	0.628	0.943	0.694	0.868
MLP_NoPre	172	1	11649	11515	0.015	0.994	0.015	1	0.029	0.507

Figure 5.7: Datos estadísticos comparativos obtenidos para todos los algoritmos con preprocesamiento y sin preprocesamiento, con la base de datos desbalanceada.

En éste caso los algoritmos que se desempeñaban bien sin preprocesamiento se mantienen en la misma linea, K-NN y PML se comportan mucho mejor por lo descrito anteriormente, pero el tiempo de ejecución es excesivo.

Tras realizar varias pruebas hemos encontrado que el atributo fnlwgt puede ser eliminado sin problema alguno, en la siguiente figura podemos observar que no tiene prácticamente peso a la hora de discriminar los datos entre una clase u otra:

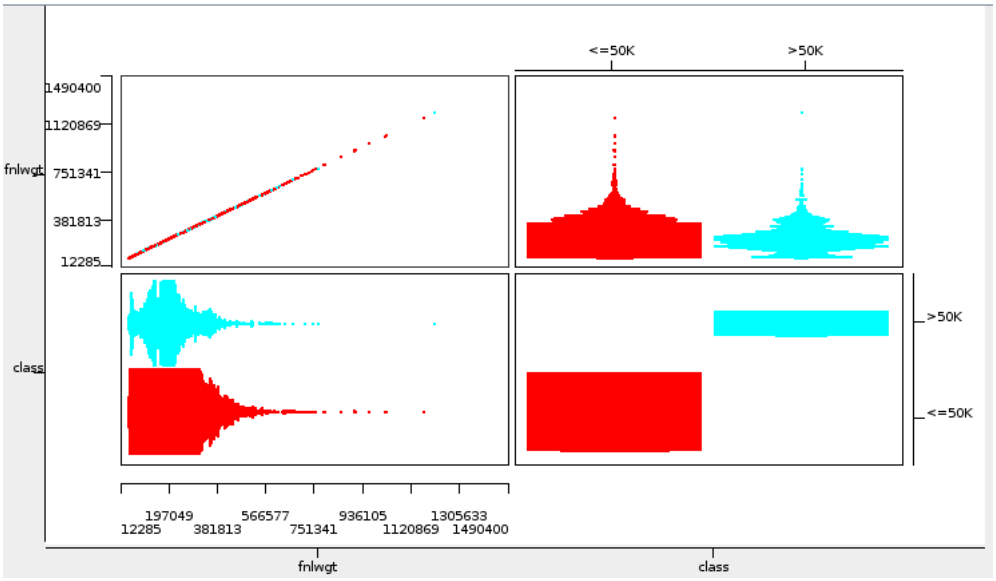

Figure 5.8: Scatter Matrix para la variable fnlwgt y class.

La variable capital-loss tambien podemos eliminarla ya que gran cantidad de tuplas tienen este valor a 0, como observamos en la siguiente figura:

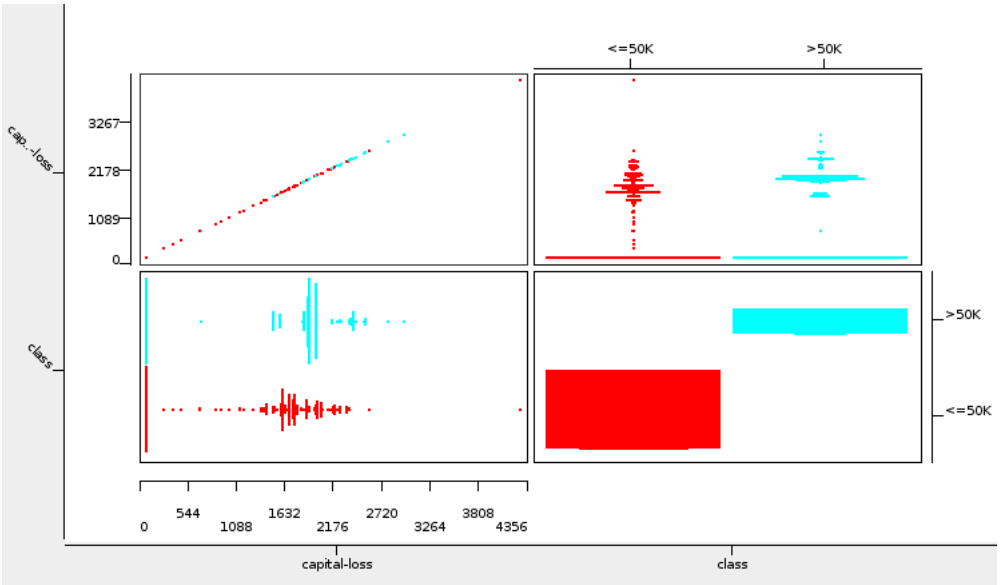

Figure 5.9: Scatter Matrix para la variable capital-loss y class.

Y realizaremos un limpidado de la base de datos, eliminando las filas que contengan valores desconocidos, dando lugar al siguiente workflow de preprocesamiento:

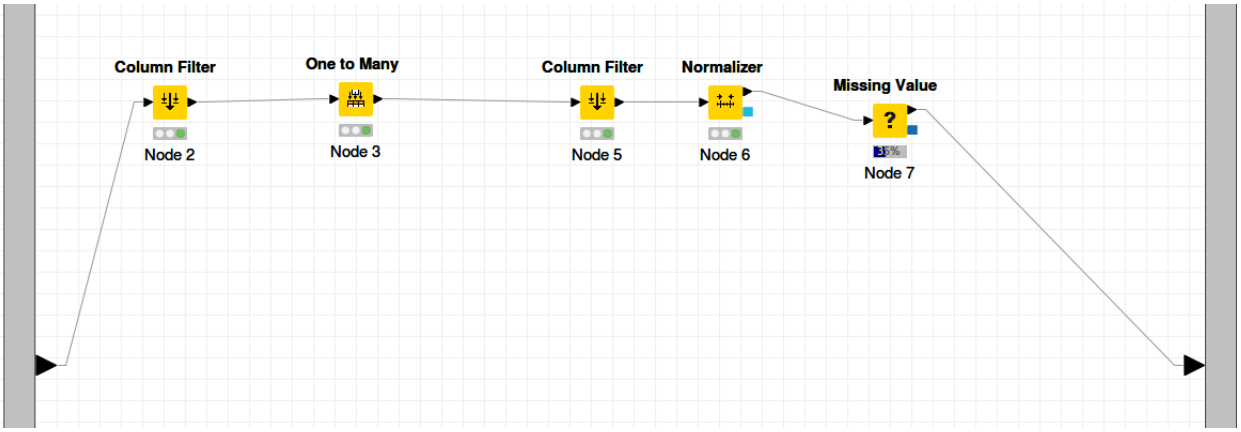

Figure 5.10: Workflow del metanodo Preprocess.

6 Interpretación de Resultados

A la vistas de los resultados y toda la experimentación realizada vamos realizar varias conclusiones sobre los algoritmos y sobre los datos en sí.

El algoritmo que mejor se ha comportado en el problema, independientemente de la preparación de datos, ha sido el Gradient Boosting, siendo este el algoritmo por el que me decantaria a la hora de resolver el problema.

En cuanto a la preparación de los datos hemos observado la importancia que tiene dependiendo del algoritmo a usar, el K-NN y el MPL necesitaban de un correcto tratamiento de los datos para su correcto funcionamiento. Y también la correcta parametrización de los algoritmos.

Y por último hemos analizado algunos atributos individualmente para poder eliminarlos del conjunto de datos sin perder capacidad de predicción en los algoritmos usados.

References

[1] http://gim.unmc.edu/dxtests/roc2.htm.

[2] http://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/InteligenciaDeNegocio//Curso17-18/adult.csv.

[3] http://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/InteligenciaDeNegocio//Curso17-18/adult.names.

[4] http://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[5] https://en.wikipedia.org/wiki/AdaBoost.

[6] https://en.wikipedia.org/wiki/C4.5_algorithm.

[7] https://en.wikipedia.org/wiki/Cross-validation_(statistics).

[8] https://en.wikipedia.org/wiki/Ensemble_learning.

[9] https://en.wikipedia.org/wiki/Gradient_boosting.

[10] https://en.wikipedia.org/wiki/Student%27s_t-test.

[11] https://en.wikipedia.org/wiki/Weka_(machine_learning).

[12] https://jeremykun.com/2015/05/18/boosting-census/.

[13] https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/.

[14] https://www.knime.com/.

[15] https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm.