Telefónica | EDUCACIÓN DIGITAL

Módulo 3:

Soluciones de Machine Learning: Aprendizaje Supervisado

Módulo 3

Soluciones de Machine Learning: Aprendizaje Supervisado

Ya hemos hablado de cómo tratar nuestros datos para que podamos usar Machine Learning y extraer las reglas o patrones que contienen. El siguiente paso es determinar qué tipo de modelos y algoritmos usar según el problema a resolver.

Los problemas que el Machine Learning es capaz de resolver se agrupan en un primer nivel en problemas de aprendizaje supervisado y no supervisado.

Aprendizaje supervisado (*Supervised Learning*)

Son aquellos problemas de Machine Learning en que la máquina aprende de un conjunto de casos o instancias previamente etiquetados por un experto o de forma semi-automática basándose en los datos, y por lo tanto necesitan de una supervisión. En este tipo de aprendizaje, el objetivo es que la máquina aprenda de los ejemplos proporcionados las reglas que nos permitirán predecir esa etiqueta para los nuevos casos que aparezcan. En el *Dataset* asociado a estos problemas, tendrá que haber un campo especial que almacene esta etiqueta. Será el *campo objetivo (objective field)*. El usuario deberá definir cuál es el campo objetivo a determinar en el momento de crear el modelo. En su defecto, se usará como objetivo el último campo categórico o numérico del *Dataset*.

Los problemas de aprendizaje supervisado son básicamente la *clasificación* y la *regresión*. Se diferencian porque en la clasificación el campo objetivo es categórico y en la regresión numérico.

En la clasificación se pretende predecir qué categoría le corresponde a una instancia dentro de una enumeración de posibles categorías. Como ejemplo de clasificación, veremos el caso de un estudio de pacientes que pueden o no tener diabetes. Usaremos un conjunto de pacientes ya diagnosticados y sus características, como sus analíticas, historia clínica y enfermedades previas, para predecir si un nuevo paciente pertenece a la clase de los que son diabéticos a la de los que no lo son.

En los problemas de regresión se quiere saber qué cantidad de alguna propiedad le corresponde a una nueva instancia. Como caso de regresión en el sector inmobiliario, podríamos querer estimar el precio de venta de una vivienda dadas sus características, como los metros cuadrados, número de habitaciones, ubicación, etc.

Ambos problemas se pueden tratar con distintos modelos. Los árboles de decisión[1] permiten resolver ambos. Hay combinaciones de árboles de decisión, como los ensembles (bagging y random decision forests)[2] que pueden mejorar los resultados obtenidos. Para el problema de clasificación también se pueden usar otro tipo de modelo llamado regresión logística[3].

En todos los casos, el objetivo del algoritmo es encontrar una función capaz de predecir para los nuevos casos. Es decir, que dadas la propiedades de un caso del que no conocemos el valor del campo objetivo, sea capaz de predecirlo lo más correctamente posible. La diferencia entre los distintos algoritmos está en la manera de generar dichas funciones y eso a veces conlleva la capacidad de obtener más o menos información de ellas. En la próxima sección explicaremos cómo funciona un árbol de decisión y qué informaciones nos aporta. El resto de modelos de clasificación y regresión mencionados quedan fuera del alcance de este curso.

Por su naturaleza, los modelos de aprendizaje supervisado pueden ser evaluados para saber qué nivel de acierto consiguen con sus predicciones. Estas evaluaciones se pueden hacer partiendo el conjunto de datos inicial en dos datasets (típicamente del 80 %-20 %). El modelo se entrena usando el 80 % de los datos y se usa el 20 % restante para predecir con él y ver en cuántas ocasiones la predicción coincide con el valor real. Este proceso de

[1] https://es.wikipedia.org/wiki/%C3%81rbol_de_decisi%C3%B3n
[2] https://es.wikipedia.org/wiki/Random_forest
[3] https://es.wikipedia.org/wiki/Regresi%C3%B3n_log%C3%ADstica

predicción y evaluación se describe en la Actividad 3.2. Podréis encontrar más información sobre la evaluación de modelos en el blog[4] y en la documentación de BigML[5].

En las próximas secciones veremos un ejemplo de solución para una problema de clasificación y cómo evaluar el modelo asociado.

3.1. Clasificación y regresión: Árboles de decisión

Los árboles de decisión son un tipo de modelo predictivo donde se utiliza un grafo con estructura de árbol para la clasificación de los datos. Cada nodo del árbol simboliza una pregunta y cada rama corresponde a una respuesta concreta a dicha pregunta (un predicado). Los nodos terminales u hojas son aquellos donde el modelo ya ha clasificado los datos y el camino desde la raíz del árbol hasta una hoja define las reglas de clasificación que cumplen los casos que han caído en dicha hoja.

Hay distintos algoritmos[6] que implementan árboles de decisión (CART, C4.5, CHAID, etc). La mayoría son binarios, es decir, de cada nodo (o pregunta) emergen dos posibles ramas (o predicados).

Veamos un ejemplo de árbol de decisión donde se está clasificando un conjunto de datos pertenecientes a pacientes que sabemos si han tenido diabetes o no. Partiendo del dataset que contiene dichos datos, crearemos un modelo con todos los parámetros de configuración por defecto como se muestra en la Figura 3.1. Esto nos genera un árbol de decisión. Previamente, habremos configurado en el dataset cuál es el campo cuyo contenido queremos predecir, que llamaremos *campo objetivo*. Esto se puede hacer desde el botón de *edición* que aparece al pasar el ratón sobre el campo o bien desde la *pantalla de configuración del modelo*. En nuestro caso, el campo que contiene esa información es el campo *diabetes*. Si no se hubiese configurado este campo, se utilizaría como tal el último campo numérico o categórico del dataset.

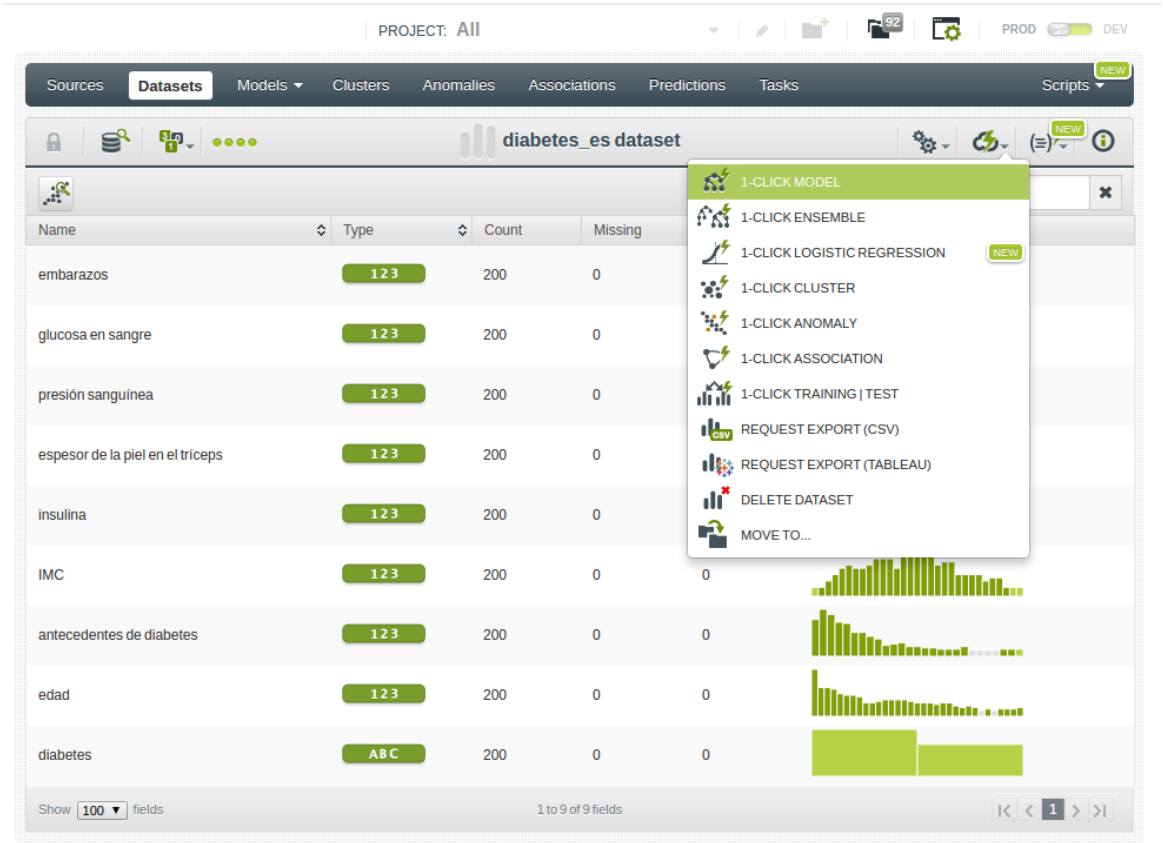

Figura 3.1: Cómo crear un *Modelo* por defecto desde un *Dataset* sobre posibles pacientes con diabetes

En la Figura 3.1 vemos que efectivamente el árbol empieza con un *nodo raíz* donde están los 200 pacientes de nuestro *Dataset*. El gráfico nos muestra los dos grupos existentes y cuántas instancias pertenecen a cada

[4] https://blog.bigml.com/2012/12/03/predicting-with-my-model-is-it-safe/
[5] https://bigml.com/documentation/dashboard/
[6] https://en.wikipedia.org/wiki/Decision_tree_learning

grupo. Vemos que la mayoría de los pacientes no tienen diabetes, y por eso la predicción del modelo en ese nodo (es decir, cuando no tenemos ninguna información sobre el paciente) es que no tiene diabetes.

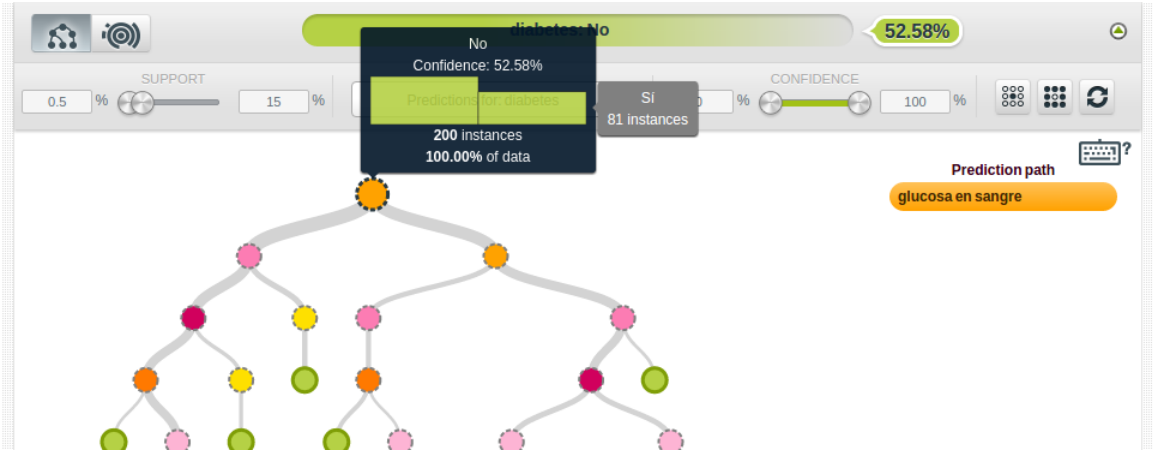

Figura 3.2: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Nodo raíz.

El objetivo del algoritmo a cada paso será separar lo mejor posible los pacientes que han tenido diabetes de los que no la tuvieron usando la información almacenada en el resto de campos. Para ello se testean las posibles opciones. Para los problemas de clasificación, se busca maximizar la ganancia de información[7] que proporcionaría utilizar cada uno de los campos y cada uno de sus valores para separar los datos. En el caso de problemas de regresión, el objetivo sería minimizar el error cuadrático medio[8]. En el ejemplo, el campo que mejor separa los datos del nodo raíz es *glucosa en sangre*, y por eso vemos que aparece en el listado de la derecha y el nodo se ha coloreado con el color asignado a dicho campo. La pregunta que mejor separa los datos es si el valor del campo *glucosa en sangre* es mayor que 123. Ese predicado es el que representa la rama que parte del nodo raíz hacia la derecha. Las instancias que cumplen dicha condición serán las que formarán parte del siguiente nodo. En el gráfico, el grosor de las ramas es proporcional al número de instancias que cumplen la condición.

En la figura Figura 3.3 vemos que 91 instancias cumplen el predicado *glucosa en sangre >123*. Entre esas instancias, la mayoría eran pacientes diabéticos. La predicción del modelo para un nuevo paciente del que sólo sepamos que tiene ese nivel de glucosa en sangre será, pues, que es diabético.

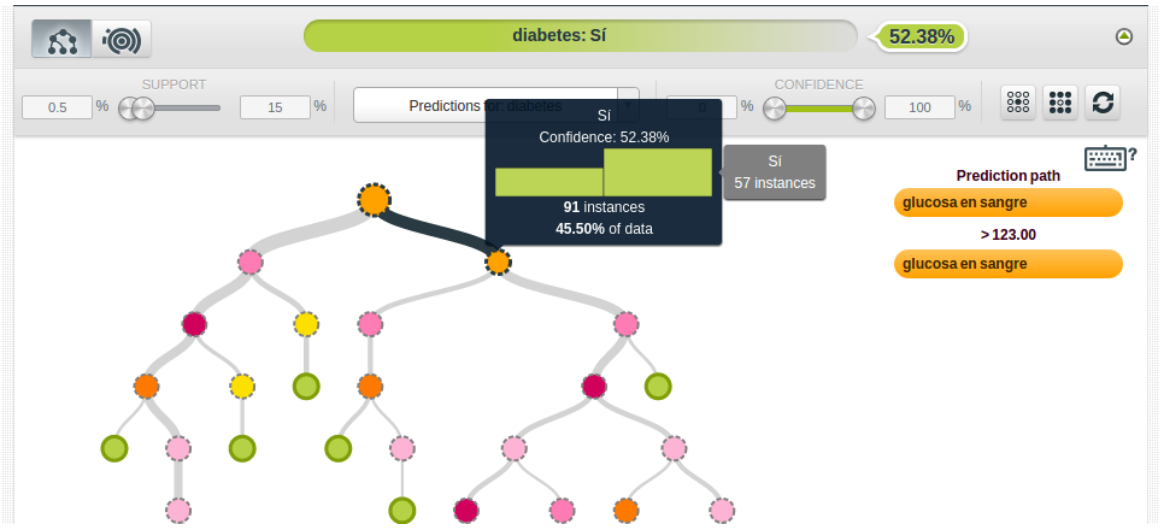

Figura 3.3: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Primer nivel para pacientes con glucosa en sangre >123.

[7] https://es.wikipedia.org/wiki/Aprendizaje_basado_en_%C3%A1rboles_de_decisi%C3%B3n#Ganancia_de_informaci.C3.B3n
[8] https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio

Si seguimos bajando por las ramas de la derecha, llegaremos a un *nodo terminal* u *hoja*. Al llegar a una hoja, el modelo no puede afinar más la separación de los datos con ninguno de los campos y valores disponibles y retorna como predicción la clase mayoritaria en ese nodo. En la Figura 3.4 vemos que las instancias de la hoja inspeccionada corresponden a pacientes con diabetes. A la derecha se muestran las reglas que cumplen todos esos pacientes. Los botones que hay en su parte inferior aparecen al apretar la tecla *mayúsculas*, y permiten descargar las reglas y crear un dataset con las instancias del nodo respectivamente. Así pues, dado un nuevo paciente que cumpla este conjunto de reglas, sabemos que la predicción del modelo será que es diabético.

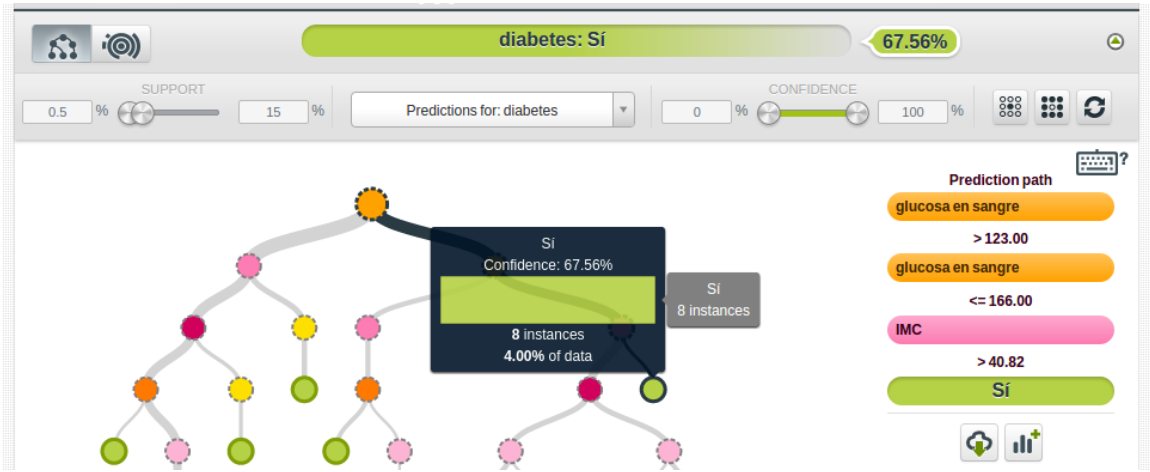

Figura 3.4: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Nodo terminal correspondiente a las reglas: 166 =>glucosa en sangre >123 e IMC (índice de masa corporal) >40.82.

Por lo tanto, la primera información que obtendremos del modelo cuando aparezca un nuevo paciente es una predicción del valor de su campo objetivo, en este caso si tiene diabetes. La predicción se obtendrá partiendo del nodo raíz y respondiendo a las preguntas que tenemos en cada nodo hasta que lleguemos a un nodo terminal o bien hasta que no tengamos información sobre el campo que se usa en la pregunta del nodo. En el ejemplo, si sólo sabemos que la *glucosa en sangre* del paciente es 130, sólo llegaremos al nodo resaltado en la Figura 3.5. La siguiente pregunta es sobre el *IMC* y no tenemos esa información. Así pues, para ese paciente el modelo predeciría que tiene diabetes.

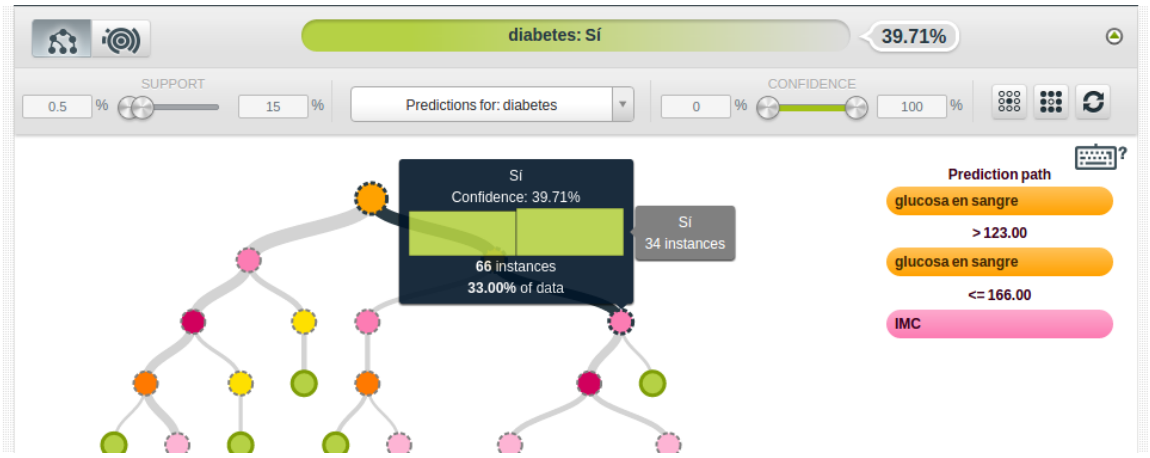

Figura 3.5: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Predicción correspondiente a las reglas: 166 =>glucosa en sangre >123.

Como vemos, la predicción viene siempre acompañada de un porcentaje de confianza (*confidence*). La confianza mide la fiabilidad que el modelo le otorga a la predicción y es más alta cuantas más instancias del nodo predicho coincidan con ella. En este caso la confianza se ha calculado basándose en el límite inferior del inter-

valo de Wilson score[9], con lo que se da una estimación pesimista. Con esta fórmula, se tiene en cuenta tanto el porcentaje de instancias del nodo predicho cuyo valor del campo objetivo coincide con la predicción, como el número total de estas.

Conviene notar que la confianza está asociada a una predicción en concreto, pero no es una buena medida de la bondad de un modelo. El modelo puede tener mucha confianza en una predicción y equivocarse. Eso ocurre, por ejemplo, cuando los modelos sufren sobreajuste[10] (*overfitting*) y por lo tanto no generalizan bien. El caso contrario, o *underfitting* también supone un problema, ya que el modelo es tan general que no captura bien las reglas existentes en los datos de entrenamiento. Por lo tanto, un buen modelo debería evitar ambos problemas y deducir los patrones existentes en los datos de entrenamiento de una forma lo suficientemente general como para que puedan ser útiles también con nuevos datos. Para medir la calidad de un modelo deberemos hacer una *evaluación*, prediciendo el valor del campo objetivo para casos en los que conocemos su valor real pero que no han sido usados en el entrenamiento del modelo. Comparando en cuántos de esos casos el valor real y la predicción coinciden o no tendremos una estimación de la bondad del modelo.

Otra información importante que se extrae del árbol de decisión es qué campos son buenos predictores del campo objetivo y su importancia a la hora de construir el modelo.

Figura 3.6: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Campos predictores y su importancia.

En la Figura 3.6 vemos los campos predictores que contribuyen más a la separación de los pacientes con y sin diabetes, encabezados por el campo *glucosa en sangre*. El porcentaje asignado se obtiene acumulando a lo largo de todo el modelo las contribuciones a la *ganancia de información* (ver Figura 3.1) de cada uno de los campos.

En algunos problemas, en realidad sólo nos interesa una de las clases de nuestro campo objetivo, y las instancias que hacen referencia a las demás se añaden para que el modelo sepa discriminarlas. En esos casos, podemos filtrar el árbol por dicha clase para ver las reglas que la predicen.

También podemos estar interesados en predicciones cuya confianza o soporte (número de instancias que secundan la predicción) supere un cierto umbral. El umbral puede ser muy variado: en diagnóstico médico se necesitan confianzas altas (superiores al 80 %) mientras que en entornos bursátiles modelos con confianzas

[9] https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval#Wilson_score_interval
[10] https://es.wikipedia.org/wiki/Sobreajuste

superiores al 50 % pueden ser considerados útiles. Para ver qué reglas nos proporcionan dichas predicciones también podemos filtrar por ambos criterios. Nos puede interesar encontrar los casos raros (confianza alta y bajo soporte) y los casos casos más frecuentes (confianza y soporte altos). Podremos ver cuales son estas predicciones accionando los filtros que vemos en la Figura 3.7.

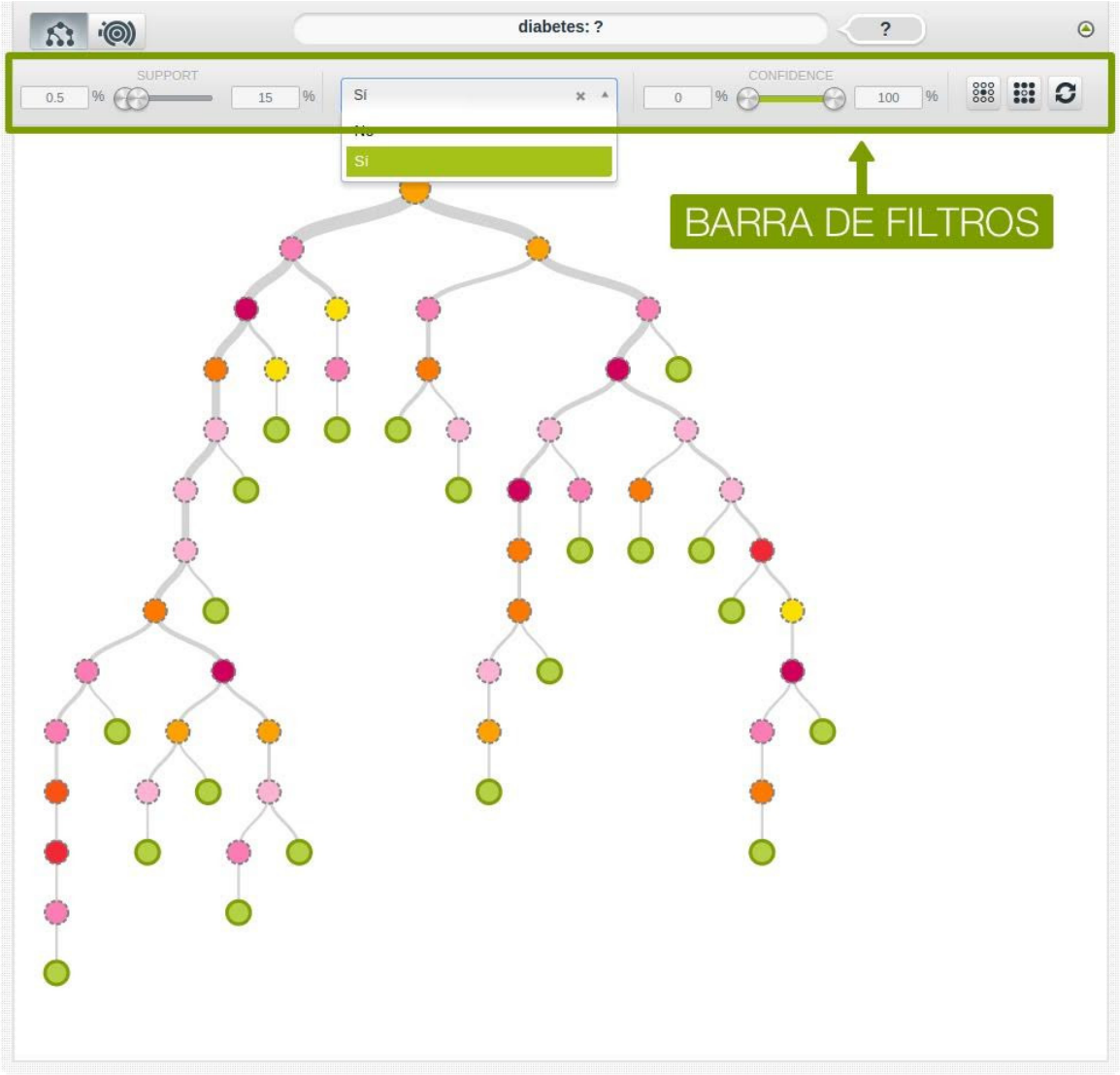

Figura 3.7: Filtrado del modelo de diabetes. Ramas correspondientes a los pacientes que tienen diabetes.

También puede interesarnos tener una visión global de cómo están distribuidas las predicciones. Para ello podemos utilizar el gráfico de *sunburst*, donde cada nodo está representado por un segmento de corona. El nodo raíz es el círculo central y se subdivide en los dos segmentos de corona que lo rodean. Cada uno de ellos a su vez se subdivide en dos segmentos correspondientes a los dos nodos de la vista de árbol, y así sucesivamente hasta llegar a segmentos terminales. La Figura 3.8 muestra dicho gráfico, que se puede colorear según la distribución de las predicciones, el nivel de confianza de cada zona o los campos que se usan en las particiones de cada nivel.

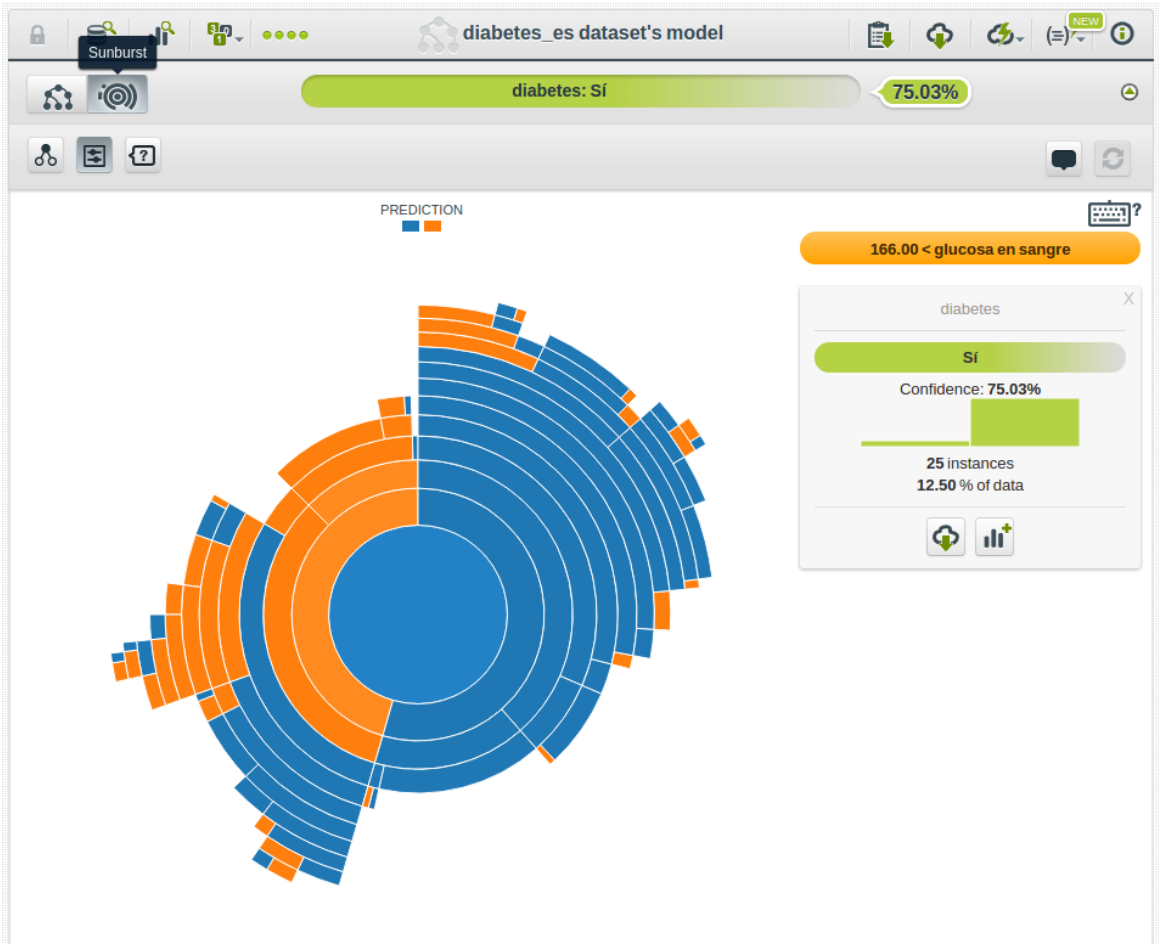

Figura 3.8: Filtrado del modelo de diabetes. Diagrama *sunburst* de las reglas de los pacientes que tienen diabetes.

3.2. Predicción y Evaluación

Como ya hemos comentado anteriormente, los modelos de aprendizaje supervisado se usan para predecir un campo objetivo, y aprenden las reglas que nos permitirán predecirlo a partir de un subconjunto de instancias de entrenamiento donde el valor de dicho campo es conocido. El hecho de disponer de los valores reales del campo para algunas instancias nos permitirá también validar la bondad de dichos modelos.

La idea sobre cómo evaluar un modelo es comparar los valores reales de las instancias en que se conoce el campo objetivo con los valores que predice el modelo para esas mismas instancias. Si ambos coinciden, el modelo estará acertando la predicción y si no habrá cometido un error. Cuantos más aciertos y menos errores, mejor será el modelo.

No obstante, debemos tener en cuenta un factor muy importante. Nunca se deben usar en este procedimiento de evaluación instancias que se hayan usado ya en la construcción del modelo. Si evaluamos un modelo con los mismos datos que usamos en su entrenamiento, lo que obtendremos es un indicador de cómo de bien se han adaptado sus reglas a dichos casos. No obstante, eso no nos permite asegurar que, cuando llegue un nuevo caso distinto de los de entrenamiento, el modelo siga dando predicciones correctas. La evaluación debe comprobar que los patrones estén adaptados a los datos de aprendizaje, pero también que sean lo bastante generales para predecir correctamente los nuevos casos que aparezcan.

El método que utilizaremos para asegurar que evaluamos correctamente es reservar un porcentaje de todas las instancias en las que el valor del campo objetivo es conocido y no usarlas en el entrenamiento del modelo. Normalmente se divide el total de instancias disponibles en un *Dataset de test*, que contiene un 20 % de las instancias, y un *Dataset de training* con el 80 % restante. El modelo se construirá con el *Dataset de training* y se evaluará con el *Dataset de test*.

Partiendo de nuestro dataset de ejemplo, vemos que el menú de acciones permite hacer dicha separación en un solo clic, tal como se muestra en la Figura 3.9.

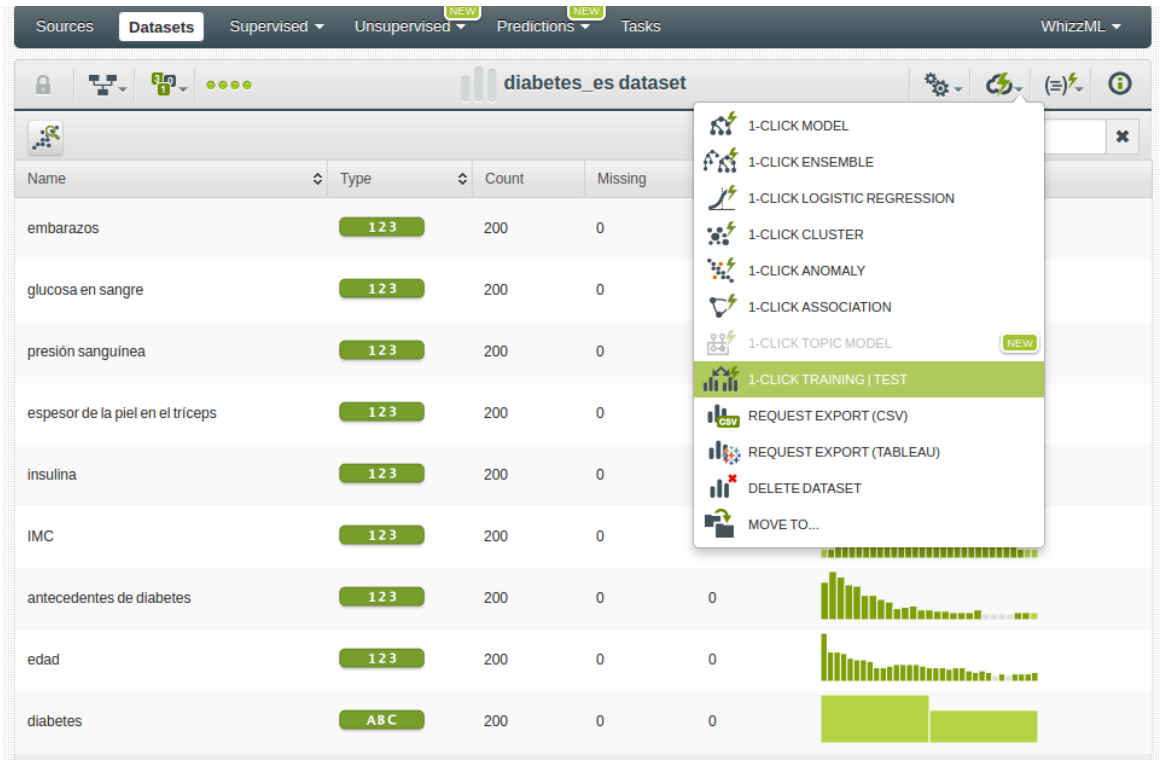

Figura 3.9: Separación de un *Dataset* en dos: 80 % de las instancias para entrenamiento y 20 % para test.

Una vez ejecutada esta acción, vemos la pantalla que nos muestra el *Dataset de training*, con lo que podremos proceder a crear el modelo con la acción *1-click model* del menú. El siguiente paso será predecir usando este modelo cada uno de los casos almacenados en el *Dataset de test*. Eso se puede hacer utilizando la opción *Batch prediction* del menú de acciones del modelo, como podemos ver en la Figura 3.10.

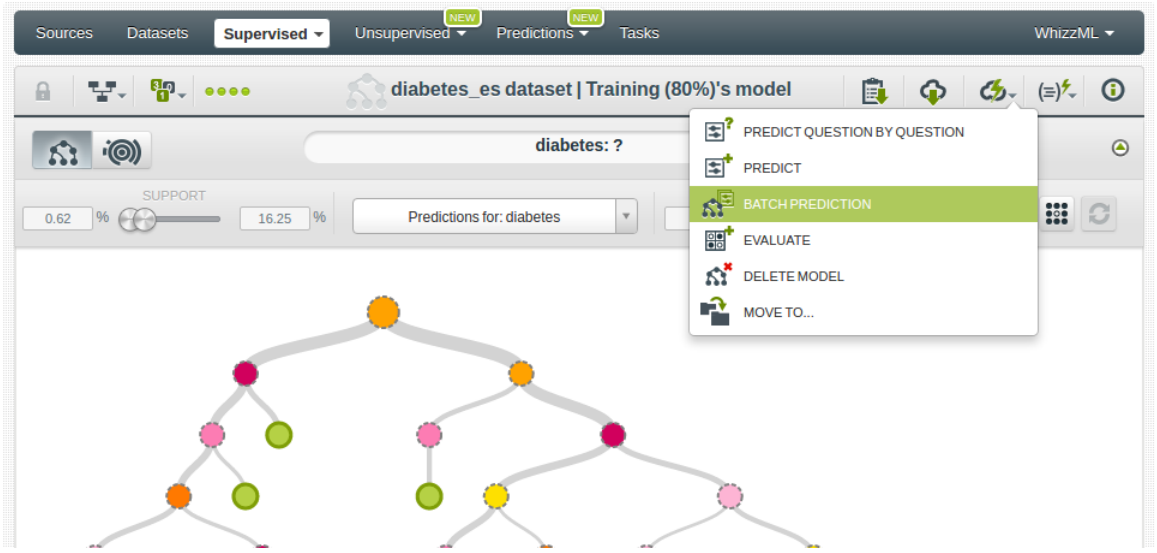

Figura 3.10: Predicción para los datos de test contenidos en un *Dataset*.

Creando predicciones para un *Dataset de test*

Con la acción *Batch Prediction* accederemos a una pantalla que nos presenta el modelo que queremos usar para predecir y nos permite escoger también el *Dataset* que contiene los datos de test, tal como se muestra en la Figura 3.11. En nuestro caso el dataset de test estará configurado por defecto, porque la plataforma detecta que hemos creado una partición de los datos para testear. El resto de opciones nos permiten configurar qué

datos obtendremos como resultado. Para poder comparar el valor real del campo *diabetes* con el valor de la predicción, cambiaremos el nombre de la columna que contendrá la predicción a *diabetes (predicción)*.

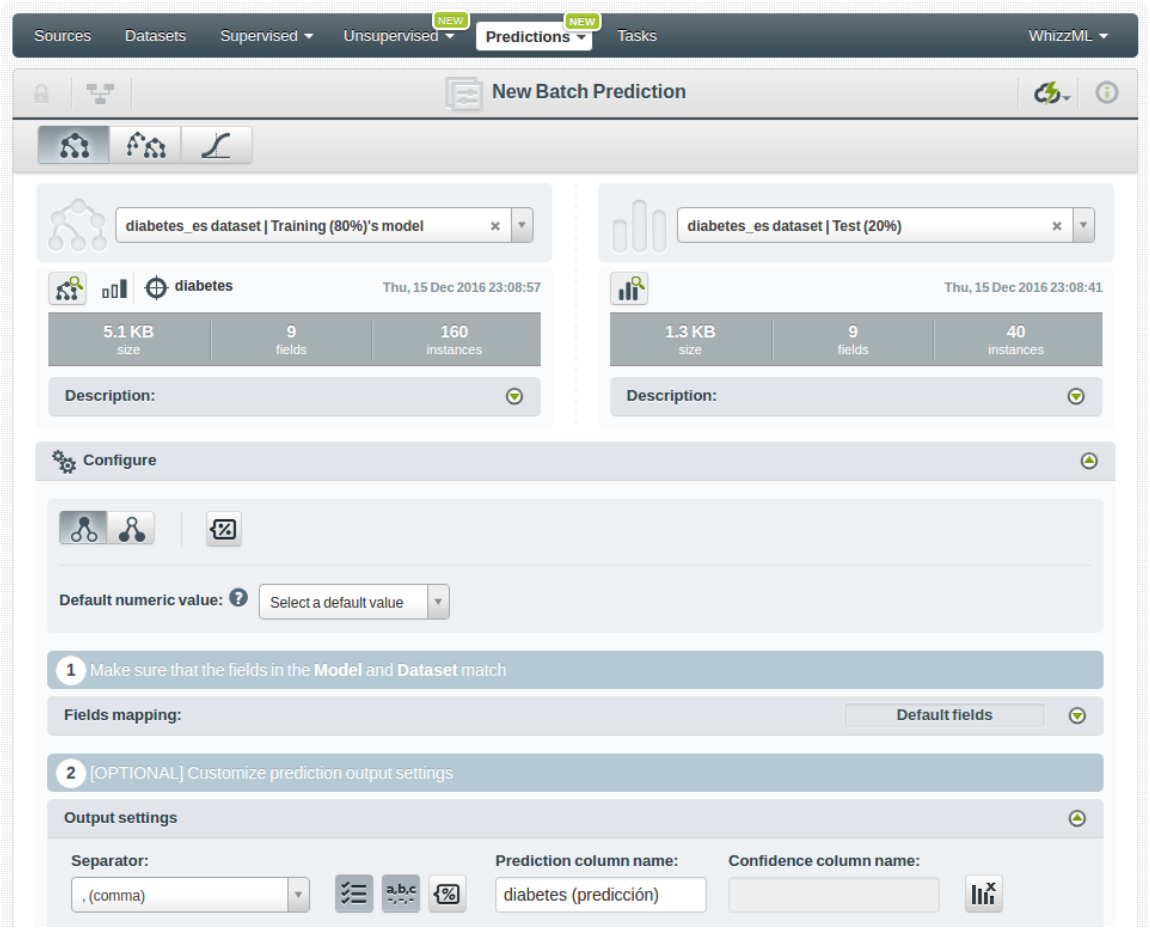

Figura 3.11: Configuración de la *Batch Prediction*: Selección del *Dataset de test* y asignación del nombre de la columna donde se almacena el valor predicho.

Al crear la predicción para todo el *Dataset de test*, aparece en pantalla una muestra de las primeras filas generadas como resultado, tal como muestra la Figura 3.12. En el ejemplo, observamos que en los primeros tres casos el campo real coincide con el predicho mientras que en los tres siguientes no y después vuelven a coincidir. Podríamos descargar el CSV y ver en cuantas ocasiones el valor predicho coincide con el real. El resumen del número de aciertos y fallos es lo que genera las diversas métricas de evaluación. En general, si no necesitamos conocer el detalle de las predicciones para cada fila del *Dataset de test* podremos obtener estas métricas creando una *Evaluación*.

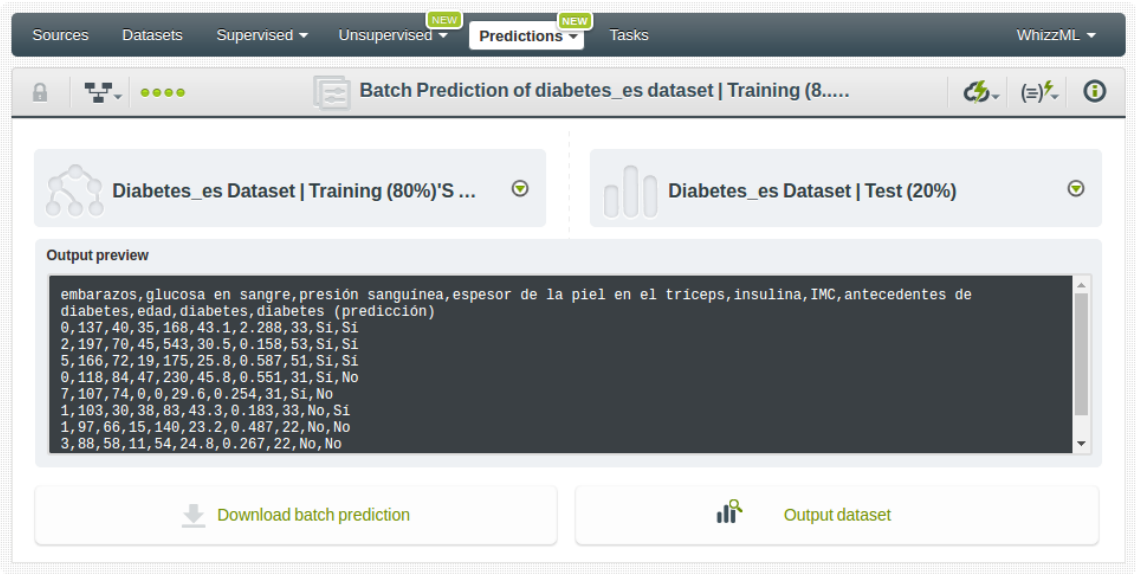

Figura 3.12: *Batch Prediction*: Pantalla de resultados donde se muestran las primeras filas de resultados. El resultado total se puede descargar en un fichero CSV o guardar en un nuevo *Dataset*.

Creando una *Evaluación*

Como en el caso de la creación de una *Batch Prediction*, el menú de acciones del modelo permite crear también una *Evaluación* usando la acción *Evaluate*, tal como muestra la Figura 3.13. Evaluar un modelo usando un *Dataset de test* consiste en realizar las predicciones que hemos visto en el apartado anterior y agrupar los fallos y aciertos en diversas métricas de evaluación que los resumen. Estas métricas se usan para comparar el rendimiento de distintos modelos entre sí.

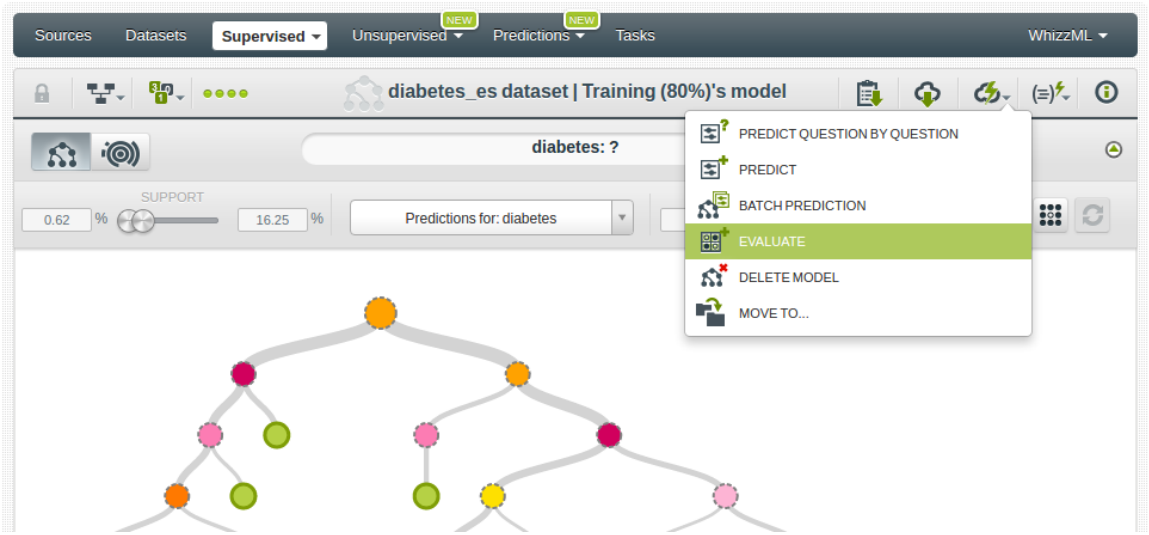

Figura 3.13: Evaluar un modelo. La acción *Evaluation* creará un recurso de este tipo que almacena las métricas de evaluación que caracterizan la bondad del modelo.

Las evaluaciones permiten comparar los resultados obtenidos con distintos tipos de modelos y también con distintas configuraciones de un mismo tipo de modelos. Por ejemplo, los árboles de decisión que hemos construido hasta ahora en nuestro ejemplo se han hecho usando la acción *1-click model*, que usa los valores por defecto de todos los parámetros configurables (véase la Figura 3.14). Al crear un árbol de decisión existen varios parámetros que se pueden elegir, como el número máximo de nodos que va a contener, o si hay que considerar más importantes algunas instancias que otras al entrenar dándoles un peso distinto. Este último caso puede ser especialmente interesante cuando el número de instancias que representan cada clase en nuestro *Dataset de entrenamiento* es muy distinto (*Datasets desbalanceados*). Cuando eso ocurre, se aconseja usar unos pesos por

instancia que permitan balancear el dataset para que todas las clases estén igual de representadas usando la opción *balance objective*. Cada modelo construido con una configuración diferente tendrá predicciones distintas para el mismo *Dataset de test*, y para comparar su acierto necesitamos usar estas métricas de evaluación, que resumen la coincidencia o no de los valores reales del campo objetivo y las predicciones de cada modelo.

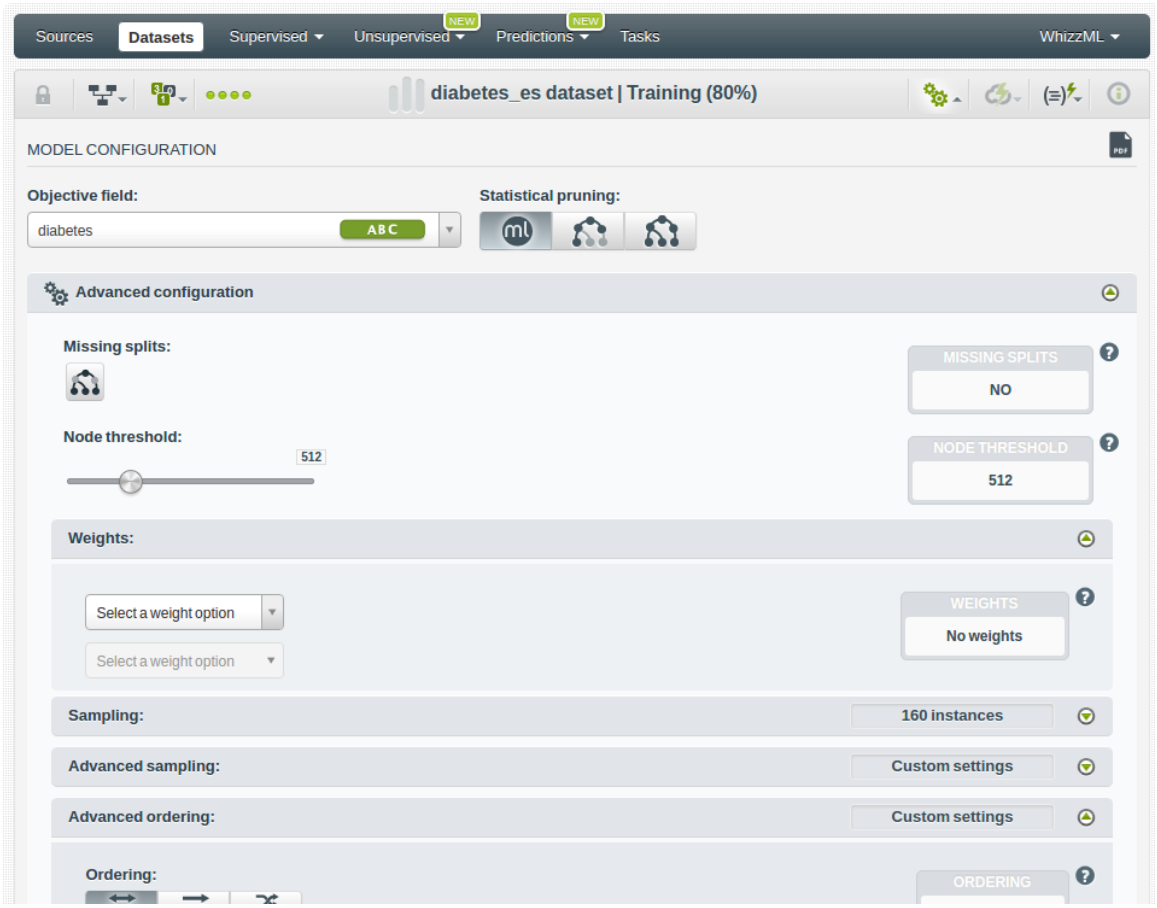

Figura 3.14: Pantalla de configuración de un árbol de decisión.

Al igual que en el caso de la *Batch Prediction*, la pantalla de evaluación nos permite elegir el modelo a evaluar y el *Dataset de test*. En nuestro caso, ambos están ya seleccionados porque la plataforma sabe que se ha partido de un *Dataset* previamente preparado a tal fin con la acción *1-click training / test*.

Al crear la evaluación, obtendremos una pantalla donde se presentan algunas de las métricas que nos permiten saber si nuestro modelo clasifica bien. La más conocida es la exactitud (*accuracy*) que mide el porcentaje de predicciones acertadas sobre el total de filas a predecir. No obstante, hay ocasiones en que la exactitud no es un buen indicador de la utilidad del modelo. El caso más claro se da cuando las clases del campo a predecir están muy desbalanceadas y solamente contamos con un pequeño número de instancias correspondientes a la clase que nos interesa predecir. Un ejemplo típico de este problema es la detección de *spam*. Los mensajes de *spam* son un pequeño porcentaje (pongamos un 5 %) cuando los comparamos con resto de correos que recibimos. Supongamos que queremos construir un modelo que nos diga si un correo es *spam* o no. Si definimos un modelo que diga que ninguno de los correos es *spam*, ese modelo tendrá una alta exactitud, porque clasificará siempre bien el 95 % de nuestros mensajes. A pesar de eso, el modelo no será bueno, porque no clasifica correctamente ninguno de los mensajes de la clase que precisamente nos interesa. Por eso, en el caso de trabajar con *Datasets* muy desbalanceados, debemos prestar atención a otras métricas, como la precisión (*precision*) y la exhaustividad (*recall*).

La precisión mide cuántas de las instancias que predecimos como pertenecientes a nuestra clase de interés (clase positiva) son realmente de esa clase. En nuestro ejemplo, la clase positiva sería la de pacientes diabéticos, con lo que la precisión nos diría que porcentaje de los pacientes que el modelo predice como diabéticos lo son realmente.

La exhaustividad mide qué porcentaje de las instancias que realmente son de la clase positiva son predichas como tal. En nuestro ejemplo, la exhaustividad sería el porcentaje de pacientes diabéticos que nuestro modelo predice como tal.

Normalmente, si construimos un modelo dando más peso a las instancias de la clase minoritaria (diabetes=Sí

en nuestro caso) en el momento de crearlo, el modelo conseguirá predecir como diabéticos más casos de entre los pacientes realmente diabéticos (mejorará la exhaustividad) pero a cambio bajará la precisión (más pacientes que no son diabéticos se predecirán como diabéticos). Para saber la calidad del modelo, será mejor pues tomar como referencia métricas como la *f-measure* o *phi* que tienen en cuenta a la vez tanto la precisión como la exhaustividad de una forma compensada.

Matriz de confusión

En realidad, todas las métricas de evaluación basan sus cálculos en cuatro variables:

- **Positivos verdaderos** (*True Positives: TP*) Los positivos verdaderos son el número de casos en que el modelo predice la clase de interés (o clase positiva) y acierta.
- **Negativos verdaderos** (*True Negatives: TN*) Los negativos verdaderos son el número de casos en que el modelo predice las clases distintas de la de interés (o clases negativas) y acierta.
- **Falsos positivos** (*False Positives: FP*) Los falsos positivos son los casos en que el modelo predice la clase positiva erróneamente.
- **Falsos negativos** (*False Negatives: FN*) Los falsos negativos son los casos en que el modelo predice las clases negativas erróneamente.

Dichas cantidades pueden presentarse en forma de tabla y forman la matriz de confusión. La pantalla de evaluación de nuestro ejemplo nos permite ver la matriz de confusión para el modelo de diabetes que habíamos construido. En la Figura 3.15 podemos ver como las filas contienen las instancias que tienen un cierto valor del campo diabetes y las columnas contienen las que se predicen con ese valor. Por ejemplo, vemos que los casos de pacientes diabéticos que son predichos como tal (**TP**) son once, mientras que los pacientes que no son diabéticos pero el modelo predice que los son (**FP**) son diez. Igualmente, se muestran los doce aciertos para los pacientes que no son diabéticos (**TN**) y los siete errores para este tipo de pacientes (**FN**).

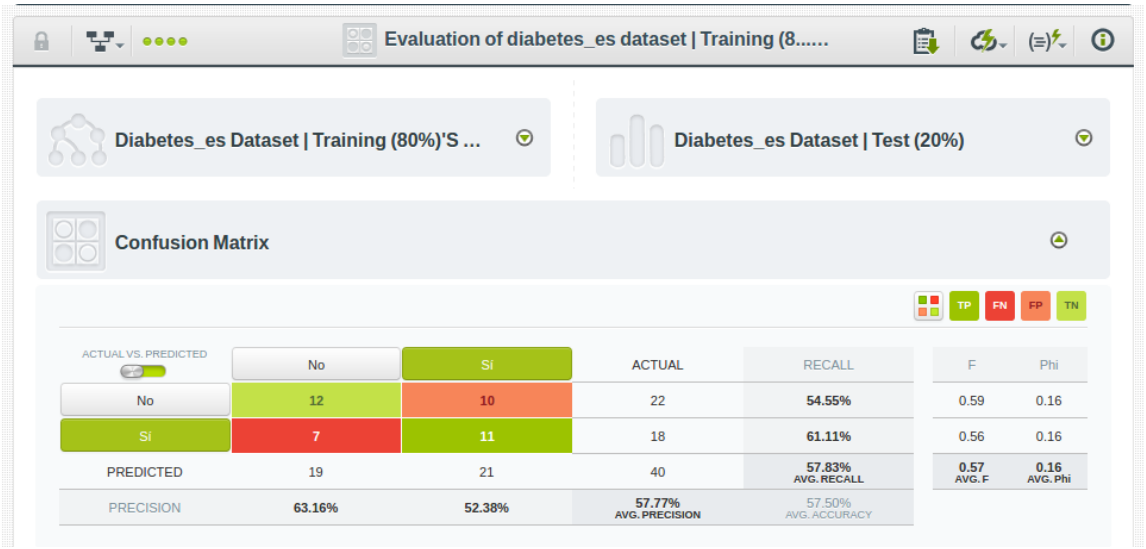

Figura 3.15: Matriz de confusión: Tabla de aciertos y errores para cada una de las clases existentes en el fichero de test y sus predicciones.

La mejor evaluación posible para saber si un modelo de Machine Learning es útil para solucionar nuestro problema es asignar un coste y un beneficio a cada una de estas cuatro cantidades (TP, TN, FP, FN). Eso es lo que se conoce como matriz de coste. Lógicamente, el coste y beneficio serán totalmente dependientes del ámbito de aplicación del modelo de Machine Learning. Los costes asociados a un error en la predicción cuando el modelo intenta diagnosticar una enfermedad grave son distintos de los que tendremos si intenta predecir el comportamiento del mercado de valores y deben ser asignados por el especialista en el dominio de aplicación del modelo. Sólo mediante este análisis podremos saber si realmente un modelo concreto nos será de más o menos utilidad.