



Introducción al MACHINE LEARNING

(con ejemplos)



1 Introducción al Machine Learning

1.1. ¿Qué es el Machine Learning?

El [aprendizaje automático](#)¹ o Machine Learning se engloba dentro de las disciplinas de la [Inteligencia Artificial](#)². Es un método científico que nos permite usar los ordenadores y otros dispositivos con capacidad computacional para que aprendan a extraer los patrones y relaciones que hay en nuestros datos por sí solos. Esos patrones se pueden usar luego para predecir comportamientos y en la toma de decisiones.

Hasta la llegada del Machine Learning, la automatización del análisis en la toma de decisiones requería siempre de un experto humano capaz de descubrir algunas reglas más o menos ajustadas. El experto, basándose en los datos, intentaba descubrir cuáles eran los distintos patrones que nos permitían resolver el problema. Formulaba un conjunto de reglas más o menos exactas y dichas reglas eran específicamente programadas por ingenieros de software. El conjunto resultaba en un modelo más o menos próximo a la realidad, que se usaba para estimar lo que iba a ocurrir en un nuevo caso o en el futuro y en eso se basaba la toma de decisiones. Este sistema de toma de decisiones presenta varios problemas:

- Escasez de expertos: A menudo nos puede ser difícil encontrar expertos formados en el dominio de nuestro problema.
- Desarrollo automatizado: Una vez los expertos han detectado las reglas, éstas deben ser automatizadas e integradas en nuestros sistemas para poder aplicarlas a grandes volúmenes de datos automáticamente. Dicho trabajo involucra a ingenieros de software, que deben entender y programar dichas reglas. En ocasiones, la transmisión de información entre el experto y el ingeniero de software da lugar a errores y requiere un tiempo considerable de implementación.
- Renovación de los modelos: La producción de una solución de este tipo acostumbra a ser lenta y costosa, y los modelos son difíciles de adaptar cuando los datos evolucionan y sus reglas cambian.
- Escalabilidad: El volumen y la complejidad de los problemas que podemos solucionar con este método es limitado.

A medida que el volumen y la complejidad de los datos han ido creciendo, se ha hecho absolutamente necesario el uso del Machine Learning. Incluso en ficheros con pocos datos, cuando el número de propiedades que se manejan empieza a crecer nos es difícil detectar los posibles patrones que los relacionan. En la [Figura 1.1](#) vemos uno de los ejemplos canónicos que se utilizan para describir cómo hacer aprendizaje automático. El fichero describe algunas propiedades de un tipo de flores, el *iris*, y el objetivo del problema es predecir la especie a la que pertenece cada ejemplar según sus características. Como se puede ver, a pesar de que el número de campos no es muy elevado, a simple vista es muy difícil detectar las reglas que determinan dicha clasificación.

¹ https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

² https://es.wikipedia.org/wiki/Inteligencia_artificial

longitud sépalo	anchura sépalo	longitud pétalo	anchura pétalo	especie
5.1	3.5	1.4	0.2	Iris-setosa
6.5	3.0	5.5	1.8	Iris-virginica
4.7	3.2	1.3	0.2	Iris-setosa
5.8	2.8	5.1	2.4	Iris-virginica
5.2	2.7	3.9	1.4	Iris-versicolor
5.4	3.9	1.7	0.4	Iris-setosa
5.7	2.8	4.5	1.3	Iris-versicolor
6.3	3.3	4.7	1.6	Iris-versicolor
4.9	2.4	3.3	1.0	Iris-versicolor
6.0	2.2	5.0	1.5	Iris-virginica
6.9	3.2	5.7	2.3	Iris-virginica
5.0	2.0	3.5	1.0	Iris-versicolor

Figura 1.1: Estructura de datos correspondiente a un ejemplo canónico de aprendizaje automatizado. Cada fila corresponde a un ejemplar de iris y sus las columnas contienen las medidas de sépalo y pétalo y la especie a la que pertenece el ejemplar. Existen reglas muy claras que nos permiten catalogar dichas flores en su especie a partir de sus medidas, pero para los humanos es difícil detectar dichas relaciones, aun cuando el número de propiedades involucradas no es muy alto, como en este caso.

Cuando la complejidad aumenta, los ordenadores superan en capacidad de análisis a los expertos humanos y evitan la necesidad de programar explícitamente soluciones a medida. De esta forma, hemos pasado de la toma de decisiones dirigida por expertos a la toma de decisiones dirigida por datos.

Los ingredientes imprescindibles para cualquier solución de Machine Learning son dos: los datos y los modelos de Machine Learning. Por eso, en el *Módulo 2* hablaremos de cómo debemos preparar los datos para poderlos usar en el Machine Learning. En el *Módulo 3* presentaremos un ejemplo de modelo de Machine Learning para la clasificación y regresión. En el *Módulo 4* trataremos otros tipos de problemas no supervisados y discutiremos los [algoritmos](#)¹ que usan para aprender. Veremos casos prácticos de cada tipo de aprendizaje y descubriremos qué modelo se ajusta mejor a cada problema y qué nuevas informaciones nos proporciona.

Por supuesto, el contenido de este curso no abarca el proceso completo necesario para integrar un sistema de aprendizaje automático en nuestra empresa. Aun así, intentaremos dar las herramientas suficientes para que podáis entender cómo funcionan algunas soluciones de Machine Learning y en qué problemas nos pueden ser de gran ayuda.

Dado que el aprendizaje automático persigue lograr que las máquinas consigan extraer patrones de los datos, el primer paso será proporcionar dichos datos en un formato que las máquinas puedan usar para aprender de ellos. En el siguiente módulo explicaremos cómo conseguirlo.

¹ https://es.wikipedia.org/wiki/Algoritmo#Tipos_de_algoritmos_seg.C3.BAn_su_funci.C3.B3n

2 Machine Learning: ¿Están nuestros datos preparados?

La realidad es que los datos que se usan en el Machine Learning normalmente proceden de distintos orígenes. Nuestros datos se almacenan en bases de datos, logs de acceso, hojas de cálculo, sistemas CRM, etc. Pongamos un ejemplo. En el caso de tener datos sobre pacientes, algunos de ellos, como la edad y su peso, pueden estar almacenados en su historial, mientras que los resultados de analíticas pueden obtenerse de un registro externo y el diagnóstico o las observaciones pueden ser introducidas en una hoja de cálculo.

Incluso en el caso de que todos los datos disponibles sobre un tema estén almacenados en un solo entorno, como una base de datos, normalmente estarán separados en varias [tablas relacionales](#)¹. Ese proceso de separación, llamado normalización, es una práctica conveniente para optimizar el almacenamiento de los datos y asegurar su mantenimiento, pero no es el adecuado como formato de entrada para los algoritmos de Machine Learning. En este módulo veremos qué estructura necesitamos dar a nuestros datos para que estén preparados para el Machine Learning y qué formatos se soportan.

2.1. Tipos de datos útiles y su formato.

El primer paso para abordar un problema de Machine Learning es, sin duda, hacer una definición clara que nos permita plantear una solución a partir de los datos.

¿Cómo definir un problema de Machine Learning?

Saber qué datos podemos usar para el aprendizaje y cómo prepararlos requiere antes que nada responder a dos preguntas.

¿Cuál es el sujeto del problema a resolver?

Supongamos que queremos saber qué usuarios de un servicio son susceptibles de querer darse de baja en el próximo mes. En este caso, el sujeto será el usuario.

Si en cambio queremos saber qué contratos pueden ser ampliados en el próximo mes, el sujeto de nuestro estudio será el contrato, dado que cada usuario puede tener más de un contrato y cada uno puede ser ampliado o no independientemente. La segunda pregunta es:

¿Cuáles son las propiedades de ese sujeto que pensamos que pueden influir en la solución?

Analicemos el sector de los seguros de salud. Podemos pensar que los usuarios que amplían sus prestaciones tienen propiedades en común. Es fácil imaginar que para predecir quién puede estar interesado en una ampliación será útil disponer de datos como la edad de la persona, los antecedentes médicos, el número de consultas telefónicas realizadas los últimos meses, el número de visitas a especialistas, las enfermedades conocidas, si hace ejercicio regular, etc. En cambio, no nos interesarán otros datos como su música preferida o el número de libros comprados en los últimos meses.

Además, tendremos que determinar si disponemos de acceso a un histórico de dichos datos. Este histórico es el que proporcionaremos al servicio de Machine Learning para que pueda basar su aprendizaje en ellos.

¹ https://es.wikipedia.org/wiki/Base_de_datos_relacional

Si también queremos que nuestro aprendizaje se pueda repetir cada cierto tiempo para asegurar que el modelo se vaya ajustando a los posibles cambios de los datos, también deberemos asegurarnos de que los datos actualizados estén disponibles periódicamente.

Así pues, tendremos que seleccionar las propiedades posiblemente relevantes, asegurar la disponibilidad de datos históricos de dichas propiedades y la periodicidad en la actualización de dichos datos.

En principio, será interesante añadir todos aquellos datos que puedan tener una relación con lo que queremos averiguar. No obstante, habrá que tener en cuenta que cada nueva propiedad añadida tendrá un coste asociado a su adquisición, almacenaje y transformación. Por eso, cuando el modelo de Machine Learning nos informe de si esa propiedad es útil o no en el aprendizaje podremos replantearnos su uso en función del análisis de [coste-beneficio](#)¹.

En el *Módulo 3* veremos que para el caso específico de los problemas de [aprendizaje supervisado](#)², como la clasificación y la regresión, existe una propiedad de la cual conocemos el valor en algunos casos y queremos predecirla para los demás. Esta propiedad concreta será lo que llamaremos el *campo objetivo* u *objective field*. Los casos en que conocemos su valor serán usados como base de aprendizaje, por lo que es especialmente importante que la definición de esta propiedad sea la correcta. En algunas ocasiones puede pasar que esta propiedad no sea exactamente uno de los campos de nuestro fichero, sino que tengamos que obtenerla mediante transformaciones sobre los datos existentes. Pasar de la definición del problema a la estructura de los datos

Una vez tengamos la respuesta a estas dos preguntas podremos crear la estructura básica que necesitaremos para aplicar el Machine Learning a nuestro problema. Consistirá en una tabla, donde cada fila contendrá la información de uno de nuestros sujetos (una *instancia*) y cada columna los valores de una de sus propiedades, susceptibles de ser útiles en el aprendizaje. Por ejemplo, en el caso de ampliación o no de contratos de servicios, cada fila contendrá toda la información sobre un contrato determinado y cada columna uno de los atributos conocidos, como el número de consultas relacionadas con ese contrato, la categoría del contrato, si ha habido reclamaciones vinculadas, etc.

A menudo, si las propiedades de nuestros datos provienen de diferentes fuentes, será necesario aplicar ciertos procesos previos para lograr esta estructura. En concreto:

DE-NORMALIZACIÓN

Las distintas propiedades de un mismo sujeto pueden estar guardadas en varias tablas relacionales. En ese caso deberemos deshacer el proceso de [normalización](#)³ para unirlas de nuevo en una sola tabla.

En la [Figura 2.1](#) podemos ver un ejemplo típico de la estructura de tablas relacionales que correspondería a una lista de reproducciones de canciones. Vemos que las propiedades de la canción están en una tabla separada, así como las de los álbumes y autores.

¹ https://es.wikipedia.org/wiki/An%C3%A1lisis_de_costo-beneficio

² https://es.wikipedia.org/wiki/Aprendizaje_supervisado

³ https://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos

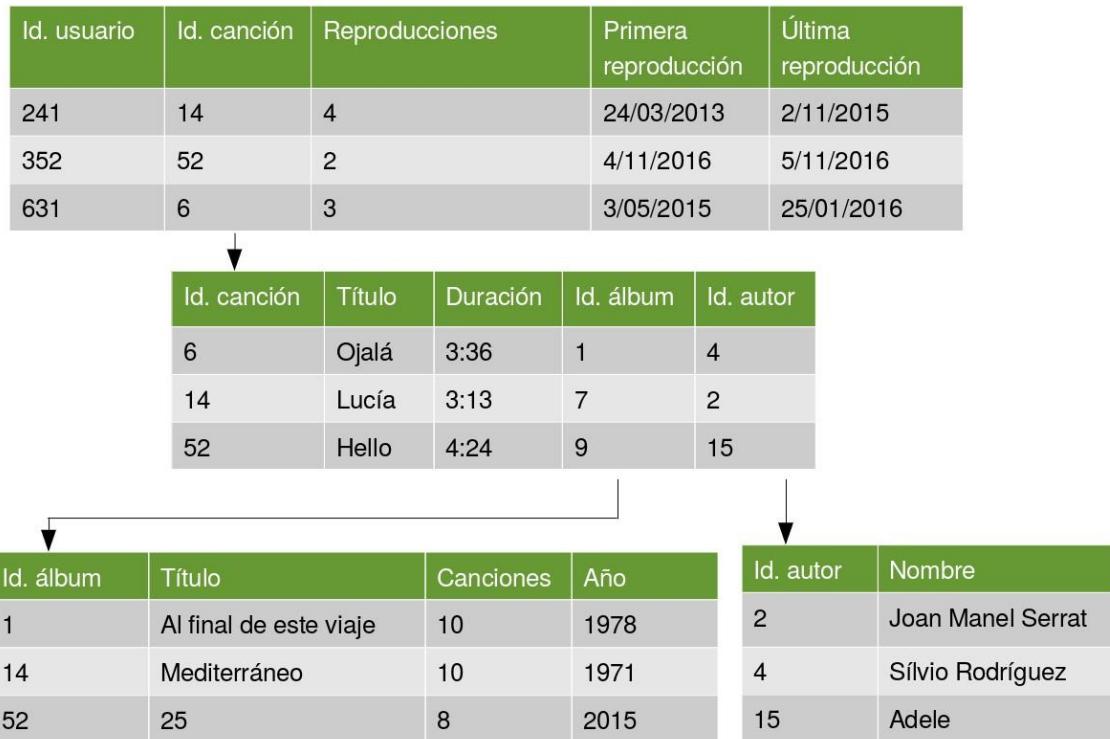


Figura 2.1: Estructura en tablas relacionales de la información sobre reproducciones de canciones. La información está normalizada para evitar redundancias

En la Figura 2.2 Vemos como de-normalizar esa separación y reunir todos los atributos que nos interesan en una sola fila por usuario.

Id. usuario	Título canción	Título álbum	Nombre autor	Reproducciones	Primera reproducción	Última reproducción
241	Lucía	Mediterráneo	Joan Manel Serrat	4	24/03/2013	2/11/2015
352	Hello	25	Adele	2	4/11/2016	5/11/2016
631	Ojalá	Al final de este viaje	Sílvio Rodríguez	3	3/05/2015	25/01/2016

Figura 2.2: Estructura preparada para el aprendizaje de la información sobre reproducciones de canciones. La información ha sido de-normalizada hasta disponer en una sola fila todas las propiedades correspondientes a un solo usuario.

AGREGACIÓN

Cuando los datos de los que disponemos son muy detallados y presentan más de una instancia para cada sujeto de nuestro problema, como por ejemplo en los logs de un servicio web, podemos necesitar agruparlos usando funciones como *contar*, *mínimo*, *máximo*, *media*, etc. El fichero preparado para el aprendizaje contendrá entonces una sola fila por sujeto y sus propiedades serán dichos agregados. En la Figura 2.3 podemos ver un ejemplo de un fichero de datos que requeriría algún tipo de agregación.

Fecha	Id. usuario
24/03/2013	241
5/05/2013	241
18/12/2013	241
3/05/2015	631
2/11/2015	241
17/12/2015	631
25/01/2016	631
4/11/2016	352
5/11/2016	352

Figura 2.3: Lista detallada de reproducciones de canciones. El fichero contiene una fila por cada reproducción.

Usando las funciones de agregación, podremos construir el fichero que vemos en la Figura 2.4. En este fichero hay una fila por cada usuario y la información detallada se ha agrupado para formar propiedades, como el número de reproducciones y las primera y última fecha de reproducción. Esta estructura estará preparada para resolver problemas como predecir el número de reproducciones para un usuario.

Id. usuario	Reproducciones	Primera reproducción	Última reproducción
241	4	24/03/2013	2/11/2015
352	2	4/11/2016	5/11/2016
631	3	3/05/2015	25/01/2016

Figura 2.4: Lista de reproducciones de canciones por usuario. El fichero contiene una fila por usuario y el detalle de las reproducciones ha sido agrupado.

PIVOTING

Similarmente a lo que vimos en el caso de la agregación, existen casos en que la información detallada incluye algún campo que nos interesaría usar también como propiedad. La Figura 2.5 muestra un ejemplo de lista de reproducciones de canciones donde el detalle incluye información sobre el soporte usado en la reproducción.

Fecha	Id. usuario	Soporte
24/03/2013	241	Tablet
5/05/2013	241	TV
18/12/2013	241	Tablet
3/05/2015	631	TV
2/11/2015	241	Smartphone
17/12/2015	631	TV
25/01/2016	631	Tablet
4/11/2016	352	Smartphone
5/11/2016	352	Smartphone

Figura 2.5: Lista detallada de reproducciones de canciones con la información sobre el aparato usado como soporte. El fichero contiene una fila por cada reproducción.

Para aprovechar esta información en el aprendizaje, deberemos incluir el detalle de cada tipo de soporte usándolo como una propiedad más en el fichero que usaremos para aprender. El proceso a realizar es transformar grupos de filas en columnas. Así, además de la columna que almacena el total de reproducciones, dispondremos de otras columnas que nos informarán del total de reproducciones por tipo de soporte.

Id. usuario	Reproducciones	Primera reproducción	Última reproducción	Tablet	TV	Smartphone
241	4	24/03/2013	2/11/2015	2	1	1
352	2	4/11/2016	5/11/2016	0	0	2
631	3	3/05/2015	25/01/2016	1	2	0

Figura 2.6: Lista de reproducciones de canciones por usuario. El fichero contiene una fila por usuario y el detalle de las reproducciones ha sido agrupado añadiendo la información del tipo de soporte como nuevas propiedades.

VENTANAS TEMPORALES

Cuando el problema de Machine Learning puede tener dependencias en la evolución temporal de nuestros datos, necesitaremos que esa información temporal se convierta también en propiedades de nuestro fichero. La manera de convertir una evolución temporal en propiedades de una tabla es crear ventanas temporales. Creamos una ventana temporal cuando resumimos en un período de tiempo el valor de nuestras propiedades. Dependiendo del intervalo temporal en que se muevan los datos, podremos definir ventanas con diferentes periodicidades: semanal, mensual, anual, etc.

Usando como ejemplo el mismo fichero representado en la Figura 2.5, podríamos crear propiedades como el total de reproducciones por año. El resultado final sería el que vemos en la Figura 2.7, donde se ha añadido una nueva propiedad por cada año documentado en nuestro fichero.

Id. usuario	Reproducciones	Tablet	TV	Smartphone	Reproducciones 2013	Reproducciones 2015	Reproducciones 2016
241	4	2	1	1	3	1	0
352	2	0	0	2	0	0	2
631	3	1	2	0	0	2	1

Figura 2.7: Lista de reproducciones de canciones por usuario. El fichero contiene una fila por usuario y el detalle de las reproducciones ha sido agrupado en ventanas temporales de periodicidad anual.

Cualquiera de estos procesos es perfectamente factible con las herramientas de sistema y las que proporcionan las bases de datos actuales. Ejemplos de comandos útiles a tal fin son: *join*, *cut*, *awk*, *sed*, *sort* y *uniq*, disponibles en los sistemas Unix¹.

EXPORTACIÓN A CSV

Una vez diseñada esta estructura en forma de tabla, procederemos a exportar los datos para generar un fichero CSV² que la replique. El formato CSV (Comma Separated Values) es un formato común de exportación que contiene solamente filas cuyos valores están separados por comas. Dicho formato no tiene información sobre el origen de esos datos o los tipos de valores que contiene cada columna. En el siguiente apartado veremos cómo esa información se puede inferir en la mayor parte de casos y podrá ser modificada cuando convenga.

EL FORMATO DE LOS DATOS

Partiendo de nuestros datos en un fichero CSV podemos empezar a usar un servicio de Machine Learning para analizarlos. El primer paso, pues, será cargar nuestros datos en el servicio que hayamos escogido. En nuestro caso usaremos la interfaz web de BigML³.

Para cargar un fichero desde nuestro ordenador basta con entrar con nuestras credenciales, previamente registradas, en BigML y arrastrar el fichero a la pantalla de listado de recursos. También podemos cargar datos desde URLs públicas, escribirlos en un editor en linea, o subirlos desde repositorios como Google Drive, Google Storage, Dropbox o MS Azure. Una vez cargados los datos veremos que se ha creado un objeto nuevo en la interfaz con el nombre del fichero subido. Llamaremos *Source* a este tipo de objetos, que almacenan las características necesarias para interpretar:

- Los campos que contiene nuestro fichero de datos. La primera fila del fichero es analizada para determinar si contiene los nombres de los campos. De no ser así, se asignan nombres automáticamente.
- El tipo de cada campo. Al lado de cada nombre vemos una imagen que nos indica el tipo de valores que contiene ese campo.

Vemos un ejemplo de *Source* en la Figura 2.8

¹ <https://www.ibm.com/developerworks/ssa/aix/library/au-unixtext/>

² <https://es.wikipedia.org/wiki/CSV>

³ <https://bigml.com>

Name	Type	Instance 1	Instance 2	Instance 3
id. paciente	1 2 3	1	2	3
fecha	DATE-TIME	04/01/16 18:43	05/01/16 00:20	06/01/16 17:17
embarazos	1 2 3	6	1	8
glucosa	1 2 3	148	85	183
presión sanguínea	1 2 3	72	66	64

Figura 2.8: Vista del objeto *Source*. En esta pantalla se muestran los campos detectados en nuestro fichero, sus tipos y el contenido de las primeras filas.

Los tipos de los distintos campos se infieren a partir de los valores detectados en las primeras filas de nuestro fichero, y pueden ser modificados si es necesario. El tipo puede ser:

Numérico: Se considera numérico aquél campo cuyos valores sólo contienen números.

Categórico: Se consideran categóricos los campos con un número limitado de valores de tipo texto que se repiten en varias instancias.

- **Fecha/hora:** Se identifican como campos de fecha/hora aquellos campos cuyo contenido encaja con alguno de los [formatos de fecha/hora](#)¹ más usuales. Los campos de este tipo son en realidad campos compuestos. Sus componentes (año, mes, día, hora, minutos, segundos) son los que pueden ser de utilidad en el aprendizaje y se añaden automáticamente al objeto *Source*. Este comportamiento puede ser cambiado en la pantalla de configuración de *Source* (Figura 2.9).
- **Items:** Se interpreta que un campo es de tipo *items* cuando contiene textos separados por algún carácter separador. La coma es el separador por defecto, con lo que los campos cuyos valores sean textos separados por comas (i.e.: *pan, jamón serrano, tomate*) serán considerados de tipo *items* automáticamente. Cada uno de estos textos entre comas se usará como una categoría o etiqueta independiente en el aprendizaje. El separador usado puede ser configurado globalmente para todos los campos *items* o bien particularmente para cada campo de este tipo en la pantalla de configuración de *Source*.
- **Texto:** Se interpreta que un campo es de este tipo cuando su contenido es de texto libre, es decir, que no está limitado a un subconjunto de etiquetas, o bien cuando el número de etiquetas distintas supera las 1.000. Estos campos se tratan al estilo de *bag of words*, separando las palabras y usando las frecuencias de las más significativas como nuevas propiedades para el aprendizaje. Existen diferentes configuraciones que pueden alterar este procedimiento: podemos elegir que no se separen las palabras, seleccionar el idioma del texto, que no se usen las *palabras vacías*, que se igualen mayúsculas y minúsculas, o que se use la [lematización](#)². Todas estas configuraciones se pueden aplicar globalmente a todos los campos de tipo texto o individualmente usando la pantalla de configuración de *Source*.

¹ <https://support.bigml.com/hc/en-us/articles/207423645-Which-date-time-formats-does-BigML-accept->

² <https://es.wikipedia.org/wiki/Lematizaci%C3%B3n>

The screenshot shows the bigml configuration interface for a CSV source named "Diabetes.csv". The interface includes sections for Locale (English (United States)), Separator (SINGLE FIELD, ; (semicolon)), Header (a,b,c), Expand date-time fields (disabled), Text Analysis (disabled), Language (Auto detect), Tokenize (All), Items Analysis (disabled), and Missing tokens (", NaN, NULL, N/A, null, -, #REF!, #VALUE!, ?, #NULL!, #NUM!, #DI'').

Figura 2.9: Pantalla de configuración del *Source*. Esta pantalla es la interfaz para poder cambiar cualquier atributo relacionado con el formato de los datos y los campos de nuestro fichero.

Cada uno de los campos detectados tendrá un identificador único, que podemos ver al pasar el ratón por encima de la etiqueta que muestra su tipo. Dicho identificador será el que se usará para hacer referencia al campo si queremos modificar cualquiera de sus propiedades, como su nombre, tipo, descripción, etc. Se pueden añadir etiquetas y descripciones a los campos para hacerlos más comprensibles.

Finalmente, la pantalla de configuración de *Source* permite modificar algunas propiedades que determinan la correcta interpretación de los valores que contiene cada campo. El *locale* determinará el carácter decimal que se usa en los campos numéricos. El separador usado en el fichero CSV cambia la estructura de columnas. El carácter que se usa como comillas determina el contenido textual. También permite definir los textos que hay que interpretar como valores ausentes y la presencia o no de la fila de cabecera con los nombres de los campos, más las opciones ya comentadas en la sección de tipos de campos.

Aunque los tipos de los campos se infieren de su contenido, existen algunos casos en que puede ser necesario cambiarlos. En algunas ocasiones los números enteros pueden ser códigos asociados a una categoría. En este caso deberíamos cambiar el tipo del campo de *numérico* a *categórico*. Otras veces podemos tener secuencias de etiquetas separadas por un carácter poco usual. Si no se ha detectado dicho separador automáticamente el campo puede aparecer como *categórico* y deberá ser cambiado a *items*, tal como aparece en la Figura 2.10. Otro caso usual es el de textos que no deben ser tratados como palabras por separado, como los nombres de países. En este caso, el cambio a realizar es de configuración del campo de texto: el valor de *tokenize* deberá ser *full terms only*.

Diabetes	Categorial	Sí	No	Sí
Observaciones	Text	antecedentes de hepatitis	N/A	N/A
Medicación	Categorial	diazepam,enalapril	N/A	simvastatina
Fecha.year	Numeric	16	16	16
Fecha.month	Categorial	April	May	June
Fecha.day-of-month	Text	1	1	1
	Items			
	YYYY-MM-DD			

Figura 2.10: Pantalla de configuración del Source. Ejemplo de cómo cambiar un campo categórico a tipo items.

Una vez veamos que nuestros datos están correctamente estructurados y formateados, pasaremos al análisis de los valores que contiene cada uno de los campos creando un *Dataset*, que describiremos en la próxima sección.

2.2. INGENIERÍA DE ATRIBUTOS ANTES DEL MODELADO

Usando la opción *1-click dataset* del menú de nuestro *Source* se crea un objeto *Dataset*. Éste es siempre el paso previo al aprendizaje, ya que en él se analiza el contenido de cada uno de los campos y se serializan los valores encontrados.

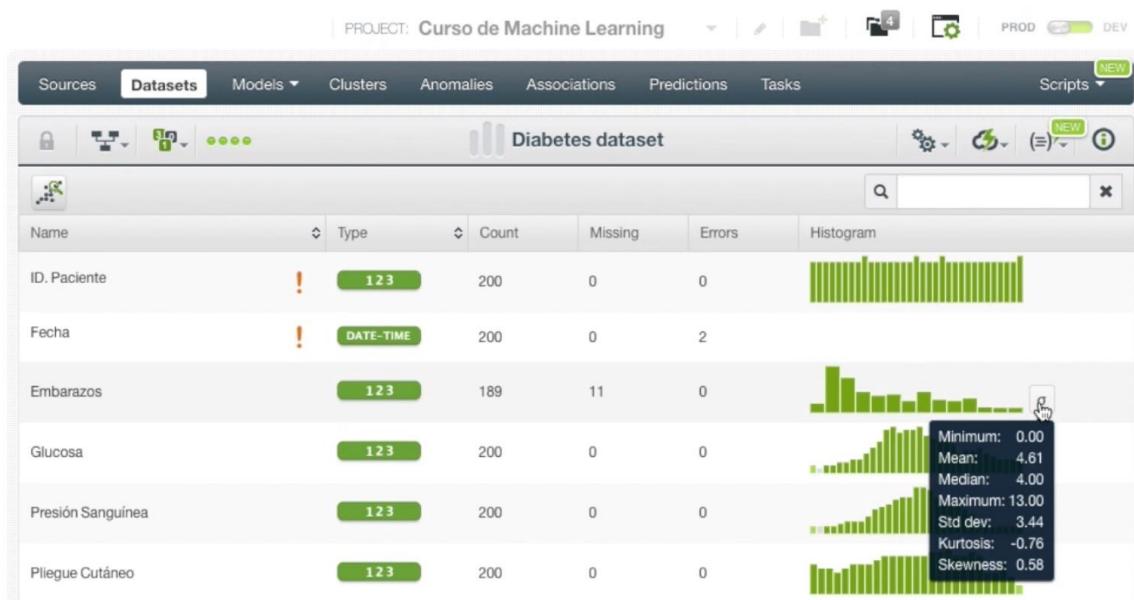


Figura 2.11: Ejemplo de pantalla de *Dataset* para el fichero de Diabetes. La pantalla de *Dataset* muestra la lista de campos y el resumen de sus valores.

En la Figura 2.11 vemos la pantalla que nos muestra la información de nuestro *Dataset*. Cada fila contiene un resumen de la información encontrada en cada uno de los campos que fueron detectados en el *Source*. Las columnas contienen:

El número de valores encontrados el número de filas del fichero donde el campo estaba vacío el número de errores (como detectar textos en un campo numérico) la distribución de dichos valores, representada gráficamente en un histograma.

Cada tipo de campo presenta un [histograma](#)¹⁰ adaptado a su naturaleza. En los campos numéricos se representa el número de instancias cuyos valores están en un rango determinado. En los categóricos, el número de instancias que

tienen una categoría asociada. En los campos de tipo *items*, la frecuencia de aparición de cada ítem. En los campos de tipo *texto*, la frecuencia de aparición de cada palabra.

Junto al histograma algunos de los campos presentan un desplegable. Para los campos numéricos el desplegable muestra las estadísticas básicas, como el mínimo, el máximo, la media y la desviación estándar. En los campos de tipo *items* o *texto* aparecen los términos más usados en un formato gráfico de [nube de palabras](#)¹¹ (o *tag cloud*), donde el tamaño de las palabras es proporcional a la frecuencia con la que aparecen.

Algunos campos pueden contener valores que no son útiles para el aprendizaje. Supongamos el caso de un campo donde se guarda el identificador de usuario. Los valores que contiene son enteros completamente distintos entre sí, y por lo tanto no habrá más de una instancia de cada valor. Este tipo de campos no será útil para el aprendizaje (*non-preferred*), y eso es lo que señala la exclamación que aparece en el primer campo de la Figura 2.11. Otro caso en que el campo aparecerá como *non-preferred* son los campos de fecha/hora, aunque en este caso el motivo es que se trata de campos compuestos. Sus componentes, como el año, mes, día, etc., son generados automáticamente y sí serán tenidos en cuenta. Lo mismo ocurrirá para campos que sólo tengan un valor constante o para campos de texto con muy pocos valores. Aunque un campo se haya marcado como *non-preferred*, se puede cambiar esta decisión usando el menú de edición que hay junto al nombre del campo. Todos aquellos campos que no estén marcados como *non-preferred* serán usados en la creación de modelos de aprendizaje.

Si queremos excluir alguno de los campos existentes en el dataset para que no se use en el proceso de modelado, podemos hacerlo desde la pantalla de configuración del modelo en cuestión. Sólo deberemos desmarcar la casilla del campo correspondiente antes de crear el modelo, tal como vemos en la figura Figura 2.12.

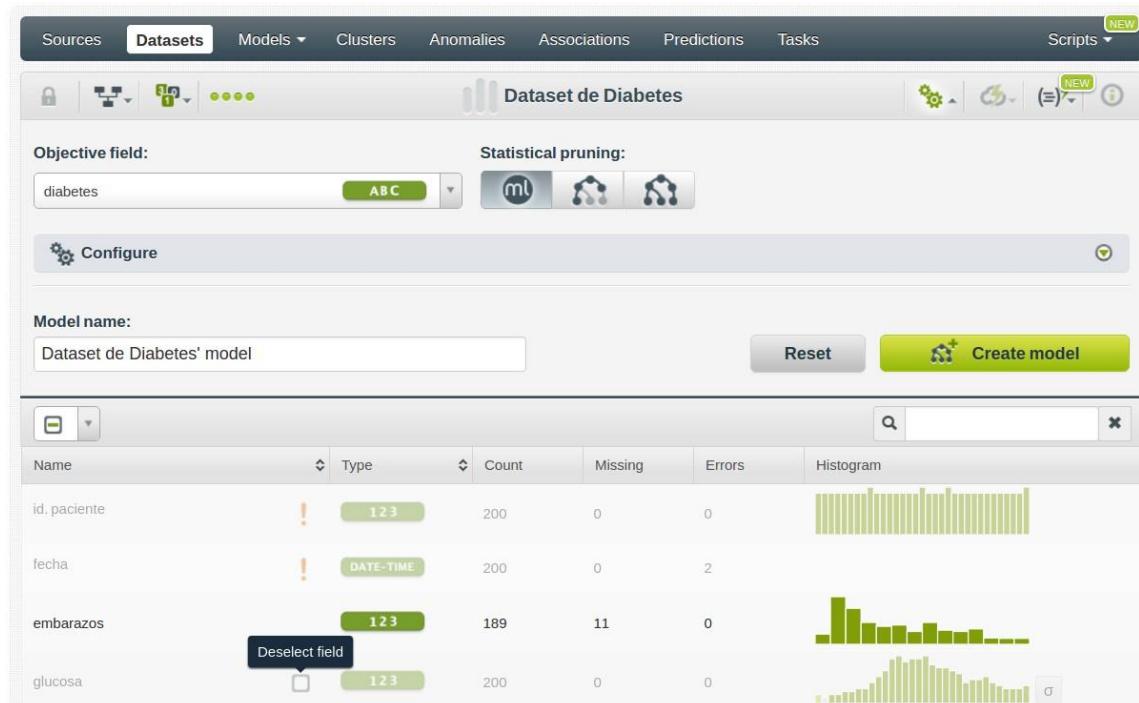


Figura 2.12: Pantalla de configuración del *Modelo*. Esta pantalla es la interfaz para poder definir cualquier parámetro necesario en la creación del *Modelo*. En este caso, el ejemplo muestra como excluir un campo del aprendizaje.

10 <https://es.wikipedia.org/wiki/Histograma>

11 https://es.wikipedia.org/wiki/Nube_de_palabras

Feature engineering para el aprendizaje

Los algoritmos que usa el Machine Learning para aprender no tienen ningún conocimiento sobre el ámbito al que pertenecen los datos ni contexto para utilizarlos. Por ejemplo, cuando un campo es numérico sólo saben que los números que contiene siguen un orden. No saben cuándo un número es par o impar, o primo, o si está dentro o fuera de un rango determinado. El algoritmo recibe el valor de un campo tal como está, y le aplica sus procedimientos para intentar aprender de él sin más información.

No obstante, las informaciones como que para agrupar un campo *edad* podría ser interesante crear tres grupos: jóvenes, adultos y mayores, o que en un campo *número* se puede distinguir entre pares y nones y eso indicará la acera de la calle donde se ubica un edificio pueden ser importantes para el modelo funcione.

Por lo tanto, en estos casos nosotros deberemos introducir esa información para ayudar al algoritmo. La forma de hacerlo es crear nuevos campos transformados a partir de los que tenemos en nuestro *Dataset* y usar el nuevo *Dataset* extendido para entrenar el modelo.

Cada nuevo campo almacenará la información sobre un predicado (por ejemplo, a qué rango de edad pertenece un individuo) o un valor calculado a partir de los datos disponibles. Es lo que se conoce como *feature engineering* o ingeniería de atributos. Siguiendo el ejemplo propuesto de los datos de diversos pacientes de diabetes, podemos ver que, para once instancias, el campo *embarazos* no tiene un valor asignado. Eso podría ocurrir por varios motivos. El más sencillo es que no se haya podido obtener dicha información, y en ese caso sería bueno llenar estos casos con el valor más frecuente, o el valor medio. También podría ser que la ausencia de valor para el campo *embarazos* fuese una señal de que esas filas corresponden a pacientes masculinos, para los cuales el campo no tiene sentido. En cualquiera de estos dos casos, deberíamos transformar nuestro dataset usando la opción *add fields to dataset* del menú de *Datasets*. En el primer supuesto, añadiendo un valor por defecto que fuese razonable. En el segundo, creando un nuevo campo que contenga información sobre si el paciente es un hombre o no.

En la Figura 2.13 se muestran algunas de las posibles transformaciones que podemos aplicar a los campos.

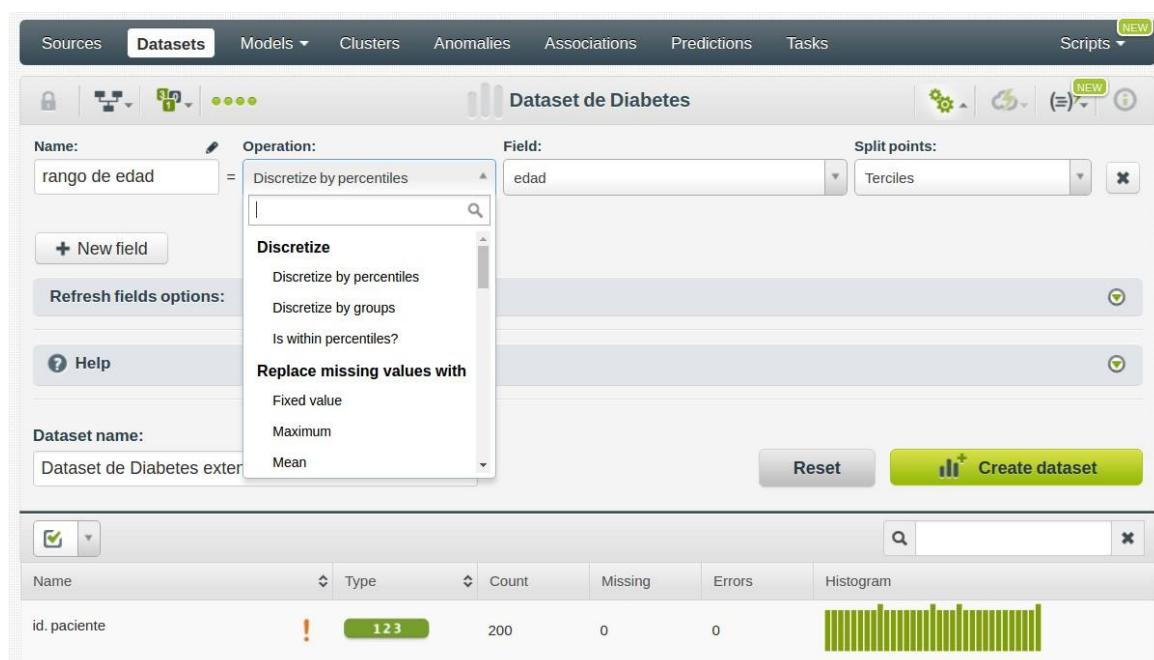


Figura 2.13: Pantalla para añadir nuevos campos a un *Dataset*. El ejemplo muestra como transformar un campo numérico a categórico usando percentiles.

Hay transformaciones usuales, como la discretización de los valores de un campo, reemplazar los valores ausentes por algún valor fijo, normalizar, o utilizar valores elegidos al azar. Así pues, transformar el campo *edad* en un campo categórico con tres rangos de edad se puede hacer usando una discretización del campo por terciles. Normalmente, una vez construidos estos campos derivados, eliminaremos el campo del cual proceden de nuestro dataset, o lo excluiremos del conjunto de campos a usar en el aprendizaje. Tener dos campos directamente dependientes como candidatos a predictores para un modelo podría afectar negativamente, ya que ambos están aportando la misma información y eso podría confundir al modelo.

En general, cualquier transformación es posible usando una *s-expression*. Existe un lenguaje para construirlas llamado [flatline](#)¹². A modo de ejemplo, podríamos construir un nuevo campo *hombre* que contuviera un *Sí* o un *No* según si el campo *embarazos* está vacío. La expresión requerida sería:

```
(if (missing? "embarazos") "Sí" "No")
```

Las expresiones pueden ser también construidas en formato JSON:

```
["if", ["missing?", "embarazos"], "Sí", "No"]
```

Hay que recordar que el *Dataset* es la base de nuestro aprendizaje. Una buena ingeniería de datos puede ser, pues, determinante en la solución de nuestro problema. Para ello, es indudablemente necesario un buen conocimiento del dominio en que vamos a aplicar el Machine Learning. Dado el gran número de transformaciones posibles, no existen todavía soluciones automatizadas que permitan determinar cuál es la mejor combinación de nuestros datos de cara al aprendizaje, aunque se está empezando a avanzar en esta dirección.

¹² <https://github.com/bigmlcom/flatline>

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 decisión¹ 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 logística³.

En todos los casos, el objetivo del algoritmo es encontrar una función capaz de predecir para los nuevos casos. Es decir, que dadas las 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.

¹ https://es.wikipedia.org/wiki/%C3%81rbol_de_decision

² https://es.wikipedia.org/wiki/Random_forest

³ https://es.wikipedia.org/wiki/Regresi%C3%B3n_log%C3%ADstica

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 predicción y evaluación se describe en la [Actividad 3.2](#). Podréis encontrar más información sobre la evaluación de modelos en el [blog¹](#) y en la [documentación de BigML²](#).

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 [algoritmos³](#) 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.

¹ <https://blog.bigml.com/2012/12/03/predicting-with-my-model-is-it-safe/>

² <https://bigml.com/documentation/dashboard/>

³ https://en.wikipedia.org/wiki/Decision_tree_learning

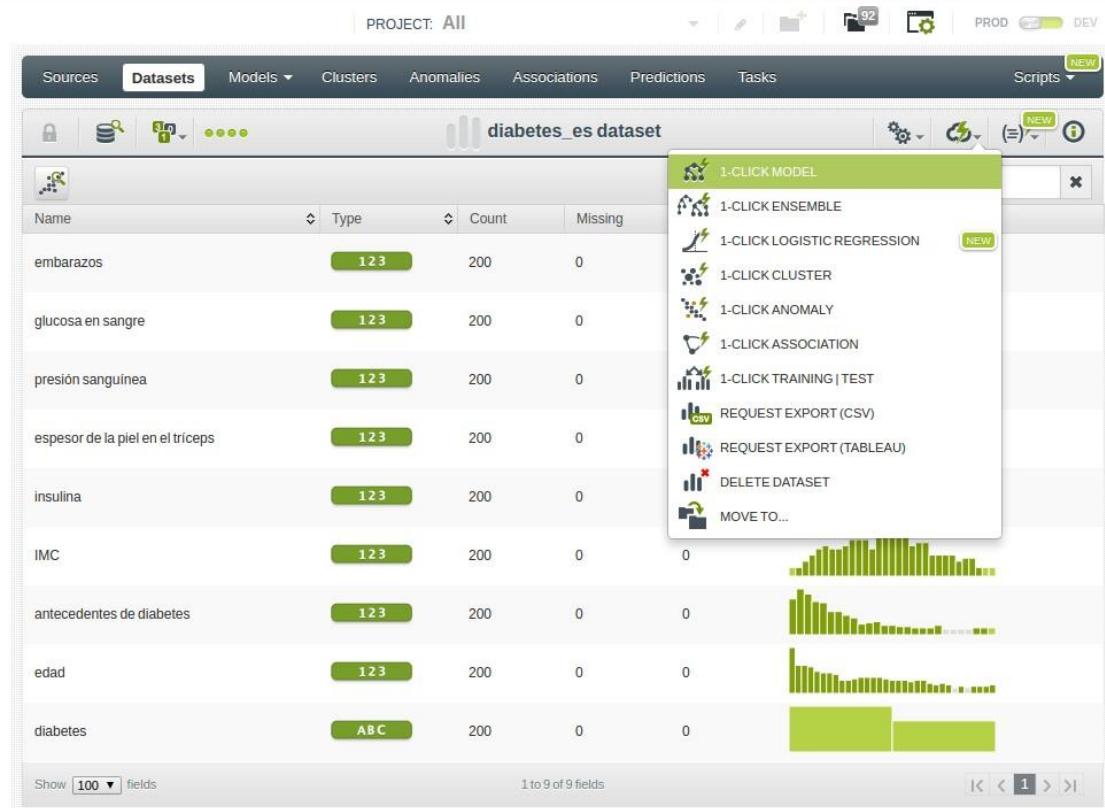


Figura 3.1: Cómo crear un *Modelo* por defecto desde un *Dataset* sobre posibles pacientes con diabetes

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 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.

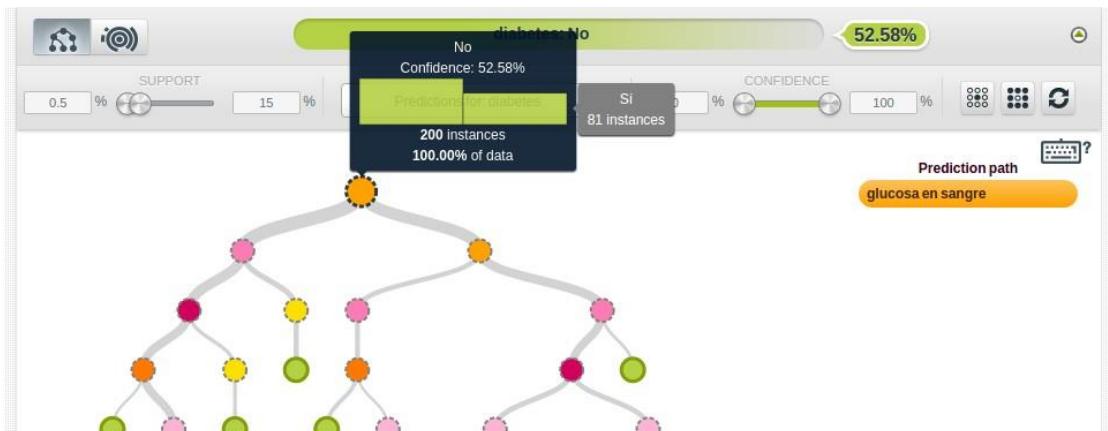


Figura 3.2: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Nodo raíz.

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](#)¹ 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](#)². 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.

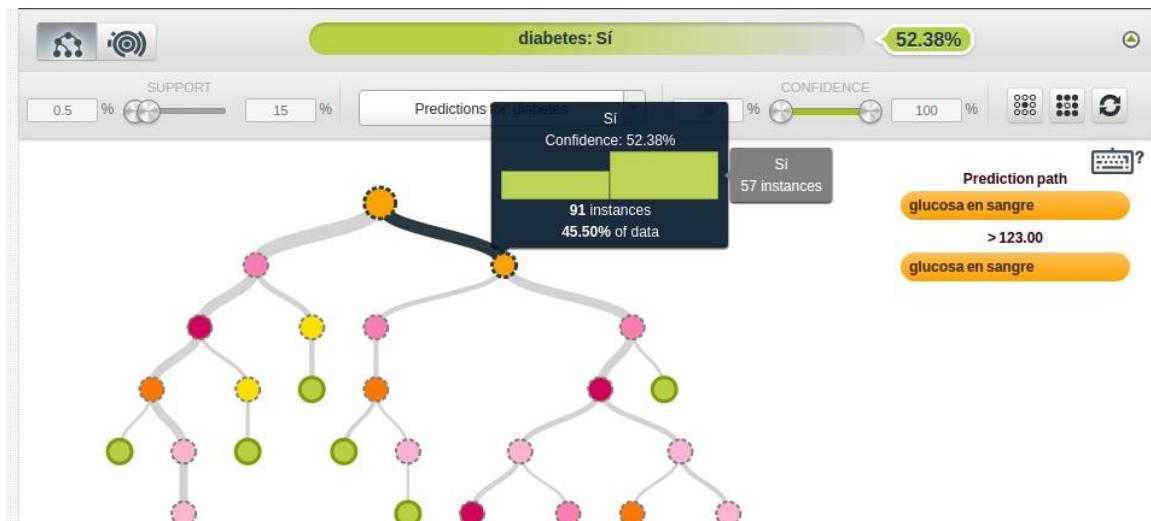


Figura 3.3: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Primer nivel para pacientes con glucosa en sangre >123.

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 *mayúscula*, 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.

¹ https://es.wikipedia.org/wiki/Aprendizaje_basado_en_%C3%A1rboles_de_decision#Ganancia_de_informaci%C3%B3n

² https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio

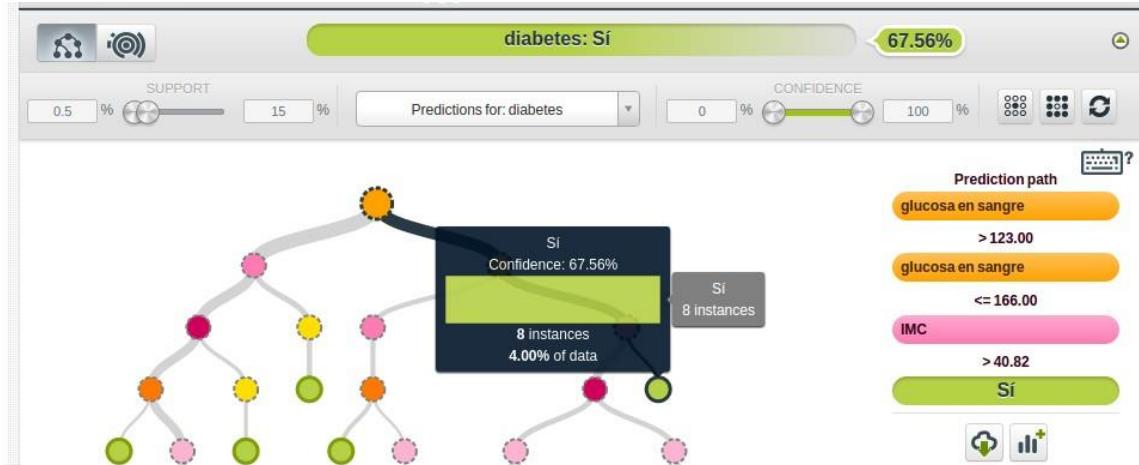


Figura 3.4: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Nodo terminal correspondiente a las reglas: $166 \Rightarrow \text{glucosa en sangre} > 123$ e IMC (índice de masa corporal) > 40.82 .

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.

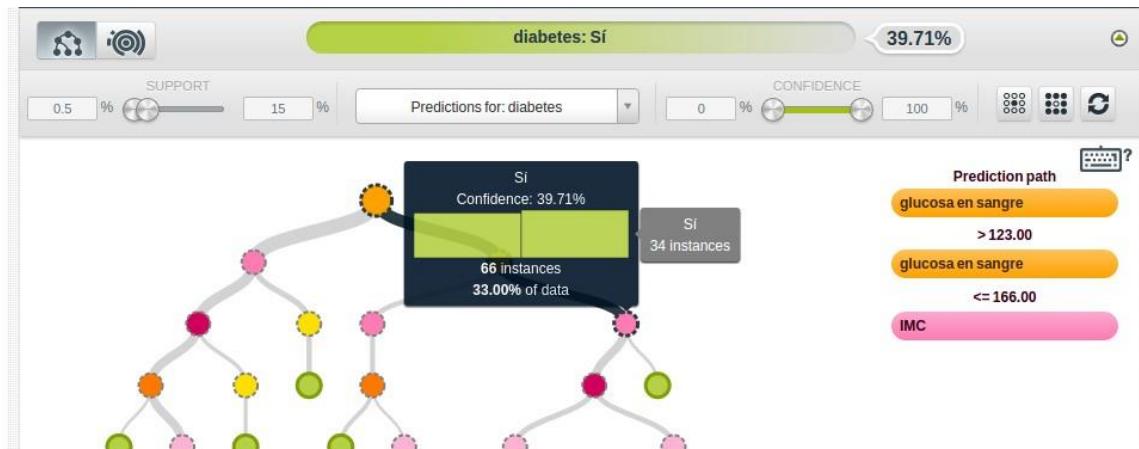


Figura 3.5: Modelo por defecto sobre pacientes con información sobre si sufrieron diabetes. Predicción correspondiente a las reglas: $166 \Rightarrow \text{glucosa en sangre} > 123$.

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 intervalo de [Wilson score⁹](#), 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 [sobreajuste¹⁰](#) (*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 discriminárlas. 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

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 más frecuentes (confianza y soporte altos). Podremos ver cuáles son estas predicciones accionando los filtros que vemos en la Figura 3.7.

9 https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval#Wilson_score_interval
 10 <https://es.wikipedia.org/wiki/Sobreajuste>

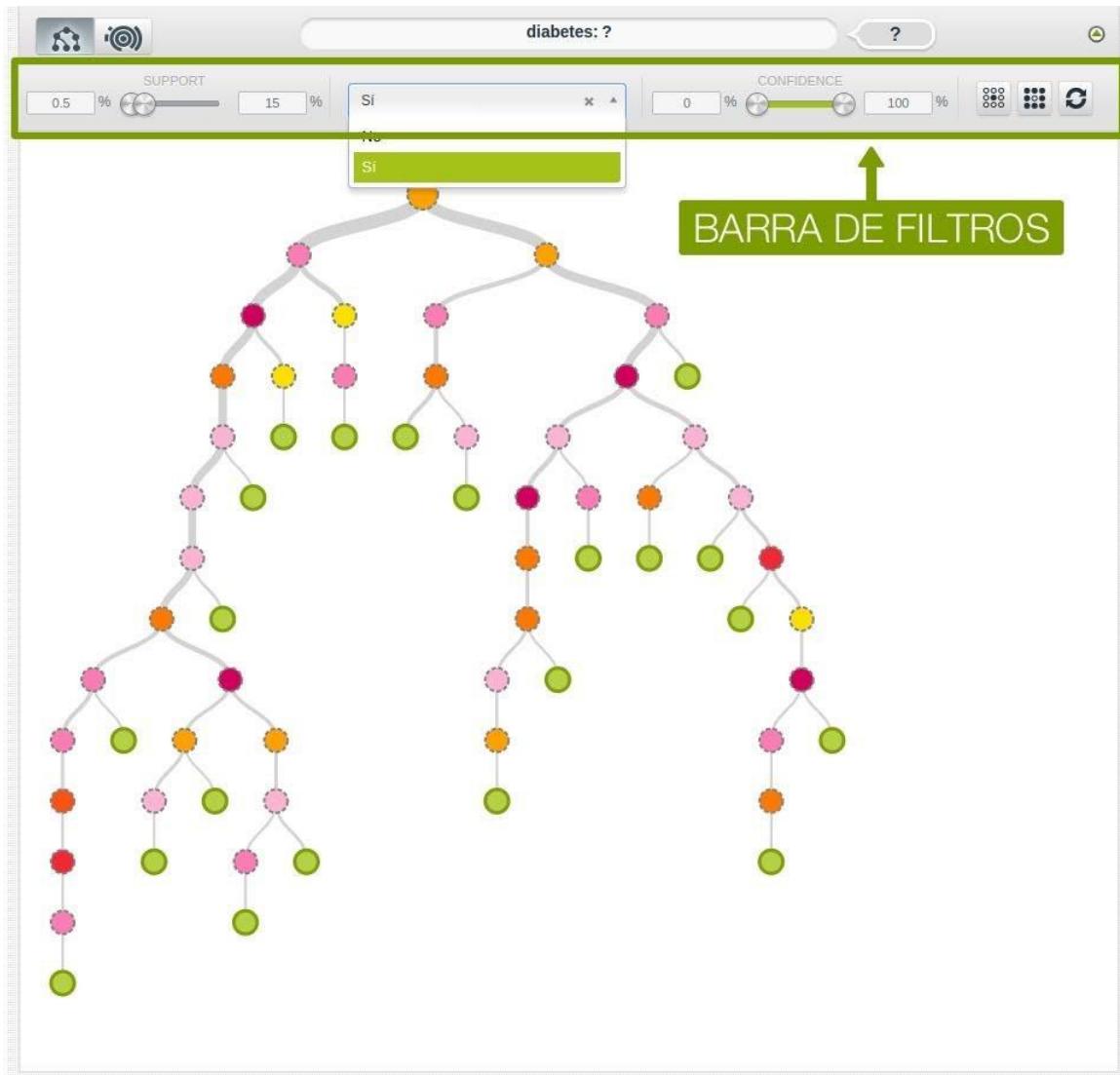


Figura 3.7: Filtrado del modelo de diabetes. Ramas correspondientes a los pacientes que tienen diabetes.

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.

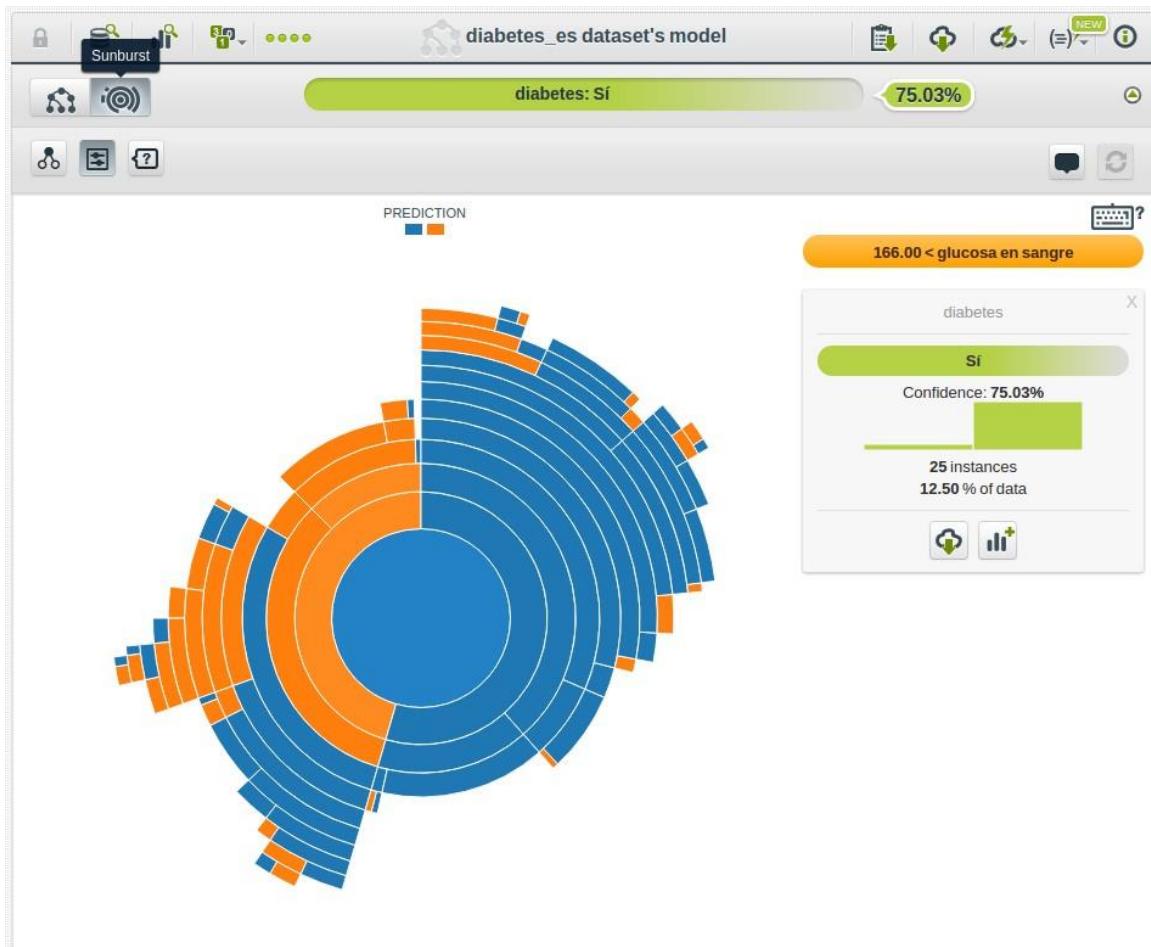


Figura 3.8: Filtrado del modelo de diabetes. Diagrama *sunburst* de las reglas de los pacientes que tienen diabetes.

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.

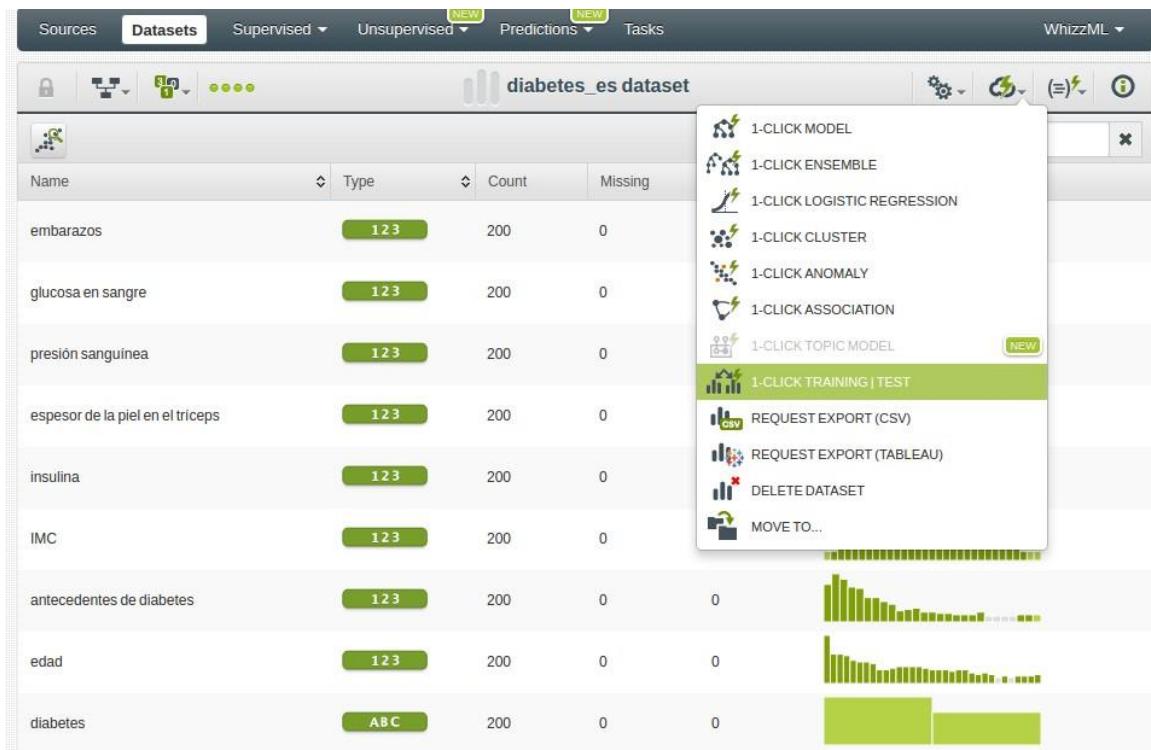


Figura 3.9: Separación de un *Dataset* en dos: 80% de las instancias para entrenamiento y 20% para test.

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.

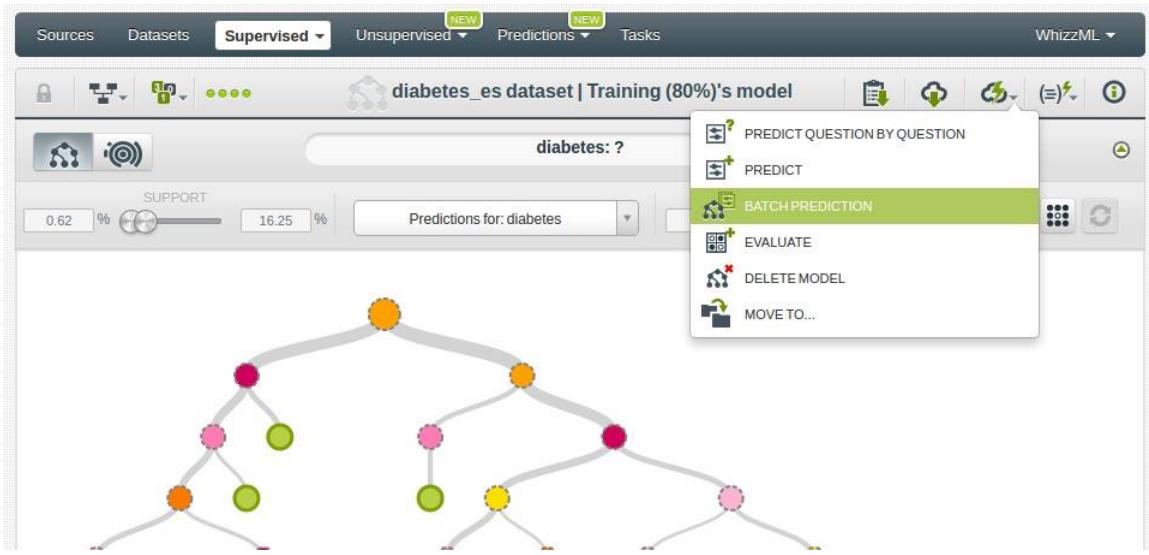


Figura 3.10: Predicción para los datos de test contenidos en un *Dataset*.

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)*.

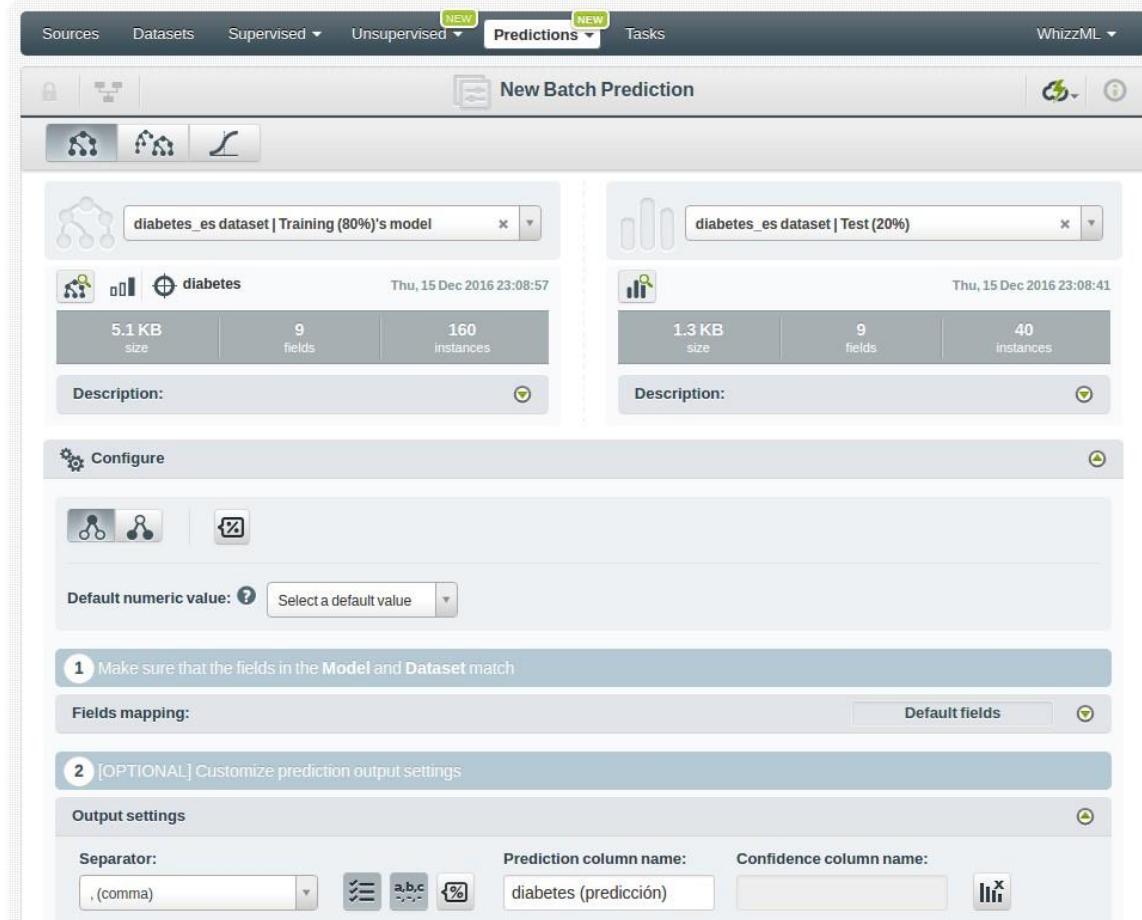


Figura 3.11: Configuración de la *Batch Prediction*: Selección del *Dataset de test* y asignación del nombre de la columna donde se almacena el valor predicho.

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.

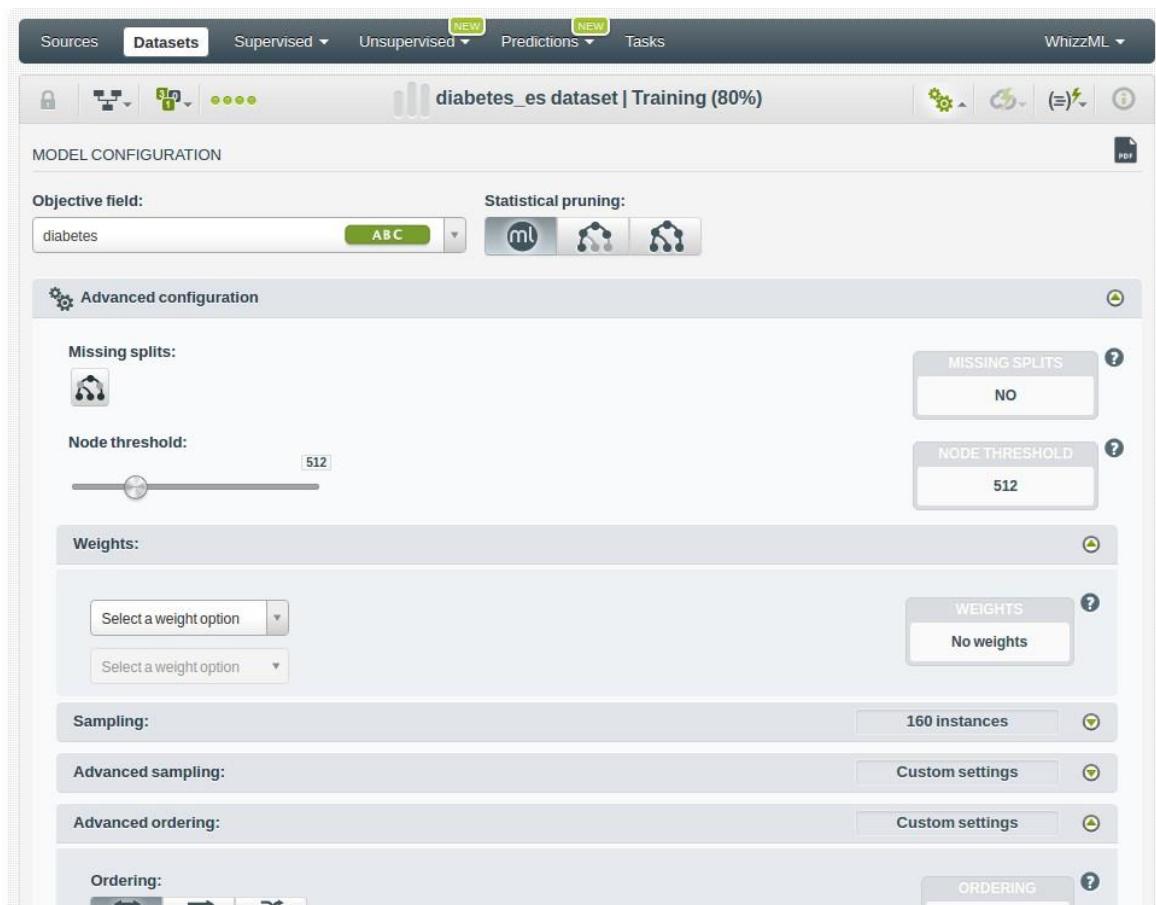


Figura 3.14: Pantalla de configuración de un árbol de decisión.

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 qué 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 métrica 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).

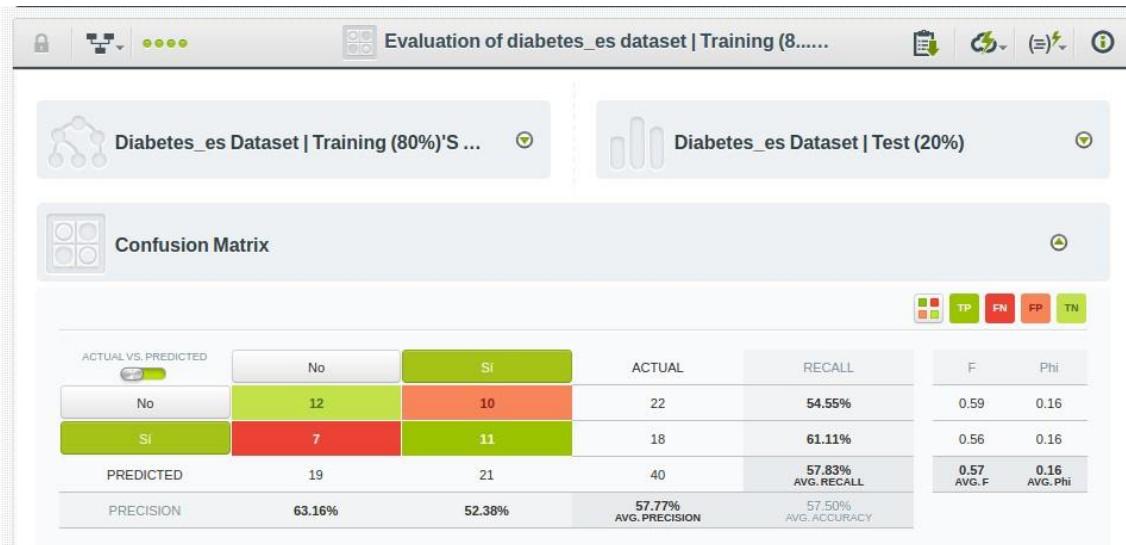


Figura 3.15: Matriz de confusión: Tabla de aciertos y errores para cada una de las clases existentes en el fichero de test y sus predicciones.

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.

4 Soluciones de Machine Learning: Aprendizaje No Supervisado

El aprendizaje no supervisado es aquél que no requiere de ningún etiquetado previo de las instancias. Se basa en los datos tal y como los recibe y su objetivo es determinar relaciones de similitud, diferencia o asociación.

APRENDIZAJE NO SUPERVISADO (*UNSUPERVISED LEARNING*)

Los modelos que se catalogan bajo este epígrafe son:

- **Clusters** Estos modelos buscan aquellas instancias que son similares entre sí y distintas de las demás para formar agrupaciones de los datos llamadas *clusters*. Dichos modelos nos permitirán predecir a qué cluster corresponde un nuevo elemento. Veremos algún ejemplo de clustering aplicado al sector inmobiliario, para segmentar la oferta de pisos y ver qué productos son similares, y al comercio, creando agrupaciones de tipos de cervezas según los gustos de nuestros clientes.
- **Detectores de anomalías** Estos modelos buscan aquellas instancias que son distintas de la tónica general del resto de datos. El ejemplo que veremos de este tipo de modelos es su aplicación al fraude en el sector de los préstamos bancarios, pero también podrían usarse para limpiar aquellas instancias de un dataset que contienen datos erróneos o muy sesgados (*outliers*).
- **Buscadores de asociaciones** Estos modelos buscan las relaciones existentes entre diversos valores de los campos proporcionados. Las relaciones, o reglas de asociación (*association rules*) son de la forma: cuando este campo tiene este valor entonces, en general, aquel campo tiene aquel valor. El ejemplo típico de este tipo de modelos es el análisis de la cesta de la compra, donde se descubren los productos que se adquieren a la vez más a menudo de lo que sería de esperar si se compraran al azar.

En general, un problema de Machine Learning necesitará aplicar combinaciones de varios tipos de modelos y refinar el proceso hasta llegar a los resultados deseados. Quizás debamos aplicar un detector de anomalías a nuestros datos para limpiar el dataset de aquellas instancias que contengan datos erróneos antes de usarlo para construir un modelo de clasificación. También podemos querer aplicar primero un clustering sobre los datos y crear modelos distintos para cada cluster para que estén más adaptados a los grupos existentes en nuestros datos. Así pues, el proceso hasta llegar a la solución del problema conllevará en general el uso de más de un tipo de modelo.

Veamos algunos ejemplos de problemas que se solucionan con algoritmos de aprendizaje no supervisado.

4.1. Segmentación: Clusters

En los problemas de segmentación o *clustering* el objetivo que se persigue es estructurar nuestros datos en grupos según su similitud. Cada grupo (o *cluster*) contendrá un número variable de instancias de nuestro dataset que se consideran similares entre sí porque los valores de los atributos que las describen lo son. A su vez, nos interesaría conseguir que los datos agrupados en un *cluster* sean claramente distintos de los que se agrupan en otro *cluster*.

¿Cómo calculamos si nuestros datos son similares?

Para poder decidir cómo se agrupan nuestros datos, los algoritmos usados se basan en la definición de una *distancia* que mide cuán similares son sus atributos. Si pensamos en atributos *numéricos*, la fórmula para calcular dicha distancia será la usual ([distancia euclídea](https://es.wikipedia.org/wiki/Distancia_euclídea)¹), de forma que cuanto menos difieren los valores de sus atributos, más similares serán las instancias y más probable será que se agrupen.

¹ https://es.wikipedia.org/wiki/Distancia_euclídea

Otro punto a tener en cuenta para calcular la distancia es cómo contribuyen los campos no numéricos. En el caso de los *categóricos*, se puede convenir que sólo contribuyen cuando las categorías de los dos casos comparados difieren. De forma similar, se debe definir cómo comparar los campos de *texto* o *items*. Para dichos campos, se analiza la frecuencia de sus términos y se comparan usando una técnica llamada [similitud coseno](#)¹ (*cosine similarity*). Cuanto más parecido es el conjunto de frecuencias de sus palabras, menos distancia habrá entre las instancias comparadas.

En cualquier caso, debemos tener en cuenta que los rangos en que se mueven los valores de diferentes atributos pueden ser muy distintos. Si estamos comparando viviendas, podemos tener atributos como el número de habitaciones, cuyos valores están por debajo de las decenas y otros como el precio, que estará en los centenares de miles. Para evitar que las diferencias en un atributo predominen sobre los demás, hay que escalar previamente los atributos y asegurar que sus valores estén dentro de un rango común. En la plataforma que usamos, estos cálculos, incluido el escalado de los atributos, se realizan automáticamente al crear los *clusters*. Nosotros podemos, sin embargo, decidir cambiar ese escalado para conseguir que un atributo concreto tenga más o menos importancia a la hora de establecer la similitud de los datos.

Finalmente, hay que comentar el caso de las instancias con campos numéricos que no tienen un valor asignado. No podemos calcular distancias cuando no tenemos dos valores numéricos a comparar. Por eso, en el caso de que algunas instancias tengan campos vacíos, o se establece un valor por defecto, como cero, el mínimo, la media, etc. o dichas instancias serán descartadas.

Tanto el escalado como los valores numéricos usados por defecto son parámetros que pueden ser configurados en la *pantalla de configuración de cluster*, tal como se muestra en la [Figura 4.11](#).

¿Sabemos el número de grupos a generar?

En ocasiones conocemos el número de clusters que queremos obtener como solución a nuestro problema de clustering. Por ejemplo, si nuestro negocio es la confección de ropa, será útil agrupar las medidas de nuestros clientes en tres clusters que nos permitan elegir las tres tallas que queremos comercializar. Otras veces en cambio, no tendremos ninguna idea a priori sobre el número de grupos a obtener y preferiremos que sea el algoritmo quien lo determine en función de la propia distribución de los datos. En cada caso usaremos un algoritmo distinto.

NÚMERO DE GRUPOS DESCONOCIDO: *G-MEANS*

Cuando no tenemos un número de grupos determinado en qué queremos organizar nuestros datos, podemos dejar que el propio algoritmo decida ese número. La decisión se toma en función de si, partiendo un grupo existente, los nuevos subgrupos contienen distribuciones normales de los datos. El algoritmo adecuado en estos casos es *g-means*.

Para entender cómo se hace la agrupación, describiremos a grandes rasgos el funcionamiento de *g-means*. Imaginemos que cada fila de nuestro fichero corresponde a un punto en un espacio donde cada coordenada es uno de sus atributos. Inicialmente partimos de un solo grupo ($k=1$) que contiene todos los puntos (es decir, todas las instancias) y lo dividimos en dos. Después se busca el centro de cada subgroupo de instancias y se unen los dos centros con una recta. Se proyecta cada uno de los puntos sobre esa recta, y se compara la distribución formada por esas proyecciones con el perfil de una Gausiana. Si encajan, consideraremos que es mejor separarlos, y aumentaremos el número de grupos (k). En caso contrario, se mantiene el grupo inicial. El proceso se aplica de nuevo a los nuevos grupos creados hasta que el número de grupos se hace estable. Esos serán los k clusters generados.

¹ https://es.wikipedia.org/wiki/Similitud_coseno

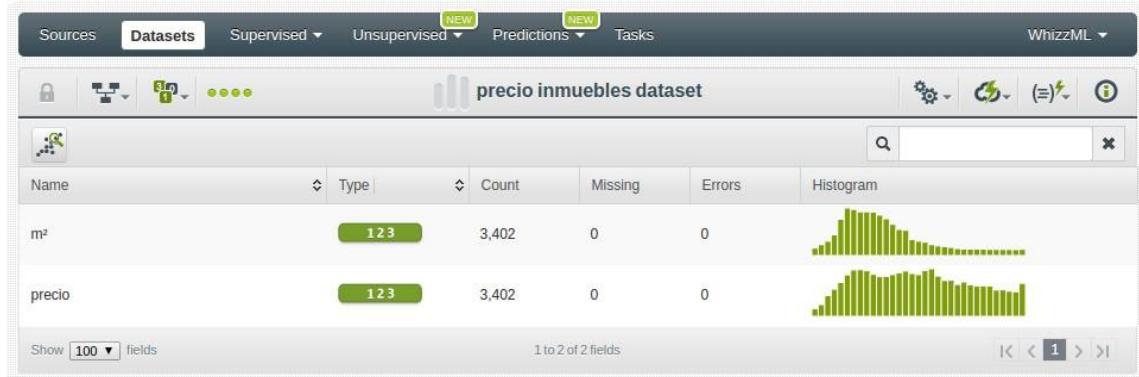


Figura 4.1: Dataset de precios de pisos. Contiene la superficie en metros cuadrados y su precio.

El algoritmo de *g-means* es el usado por defecto cuando creamos un *Cluster* desde un *Dataset*. Como ejemplo sencillo podemos ver cómo se agrupan los datos sobre inmuebles según su superficie y precio. Partimos de un *Dataset* que contiene esos datos, como el de la Figura 4.1 y crearemos un *Cluster* con la opción *1-click Cluster* del menú de acciones del *Dataset* tal como vemos en la Figura 4.2.



Figura 4.2: Creación de un *Cluster* en un solo clic. El algoritmo usado es *g-means* y todas las opciones son por defecto.

La representación gráfica de los *clusters* nos presenta cada uno de ellos como un círculo coloreado, tal y como muestra la Figura 4.3. El tamaño de cada círculo es proporcional al número de instancias que pertenecen a ese *cluster*. Para describir un *cluster* usaremos su centro (o *centroid*). Vemos que, al mover el ratón por encima de cada círculo, se muestra en pantalla la información del *centroid* correspondiente. Sus propiedades son el promedio de las propiedades de las instancias agrupadas en el *cluster*. Si además presionamos la tecla *mayúsculas*, aparecerá el botón situado bajo la tabla de propiedades del *centroid* que nos permite generar un nuevo dataset con las instancias agrupadas en ese *cluster*.

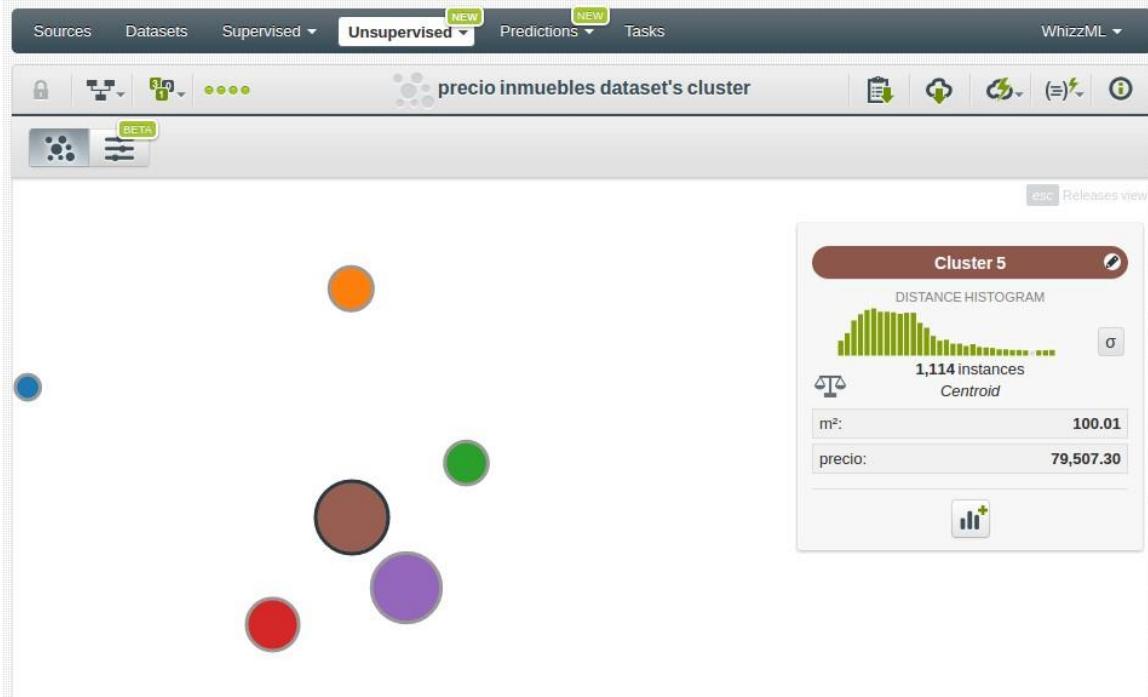


Figura 4.3: Visualización de los clusters creados con *g-means*. Cada círculo simboliza un grupo o *cluster*.

Otro elemento importante a tener en cuenta es la distancia existente entre los diversos *centroids*, ya que nos dará una idea de cuán distintos son los *clusters* generados. En nuestro gráfico, los círculos más alejados del *cluster* central representan los *clusters* con instancias más diferentes de las agrupadas en el *cluster* central y los más cercanos los más similares. Haciendo un clic sobre cualquier *cluster* del gráfico, éste pasará a ser el central y los demás se redistribuirán para mostrar sus distancias con relación a él. Las distancias entre los demás *clusters*, excluyendo el central, están adaptadas para evitar que los círculos se superpongan, y no tienen ninguna traducción en términos de las propiedades de dichos *clusters*.

En el ejemplo que hemos construido, vemos claramente que el *cluster* elegido como centro en este gráfico es el que se ha etiquetado como *Cluster 5*. Contiene 1114 instancias cuya distancia al *centroid* se encuentra distribuida según muestra el histograma. La lista de campos del *centroid* muestra los usados en el cálculo de la distancia: la superficie en metros cuadrados y el precio. La media de estos valores para todas las instancias que han sido agrupadas en este *cluster* es la que vemos asociada al *centroid*. El símbolo de la balanza que hay sobre la tabla de valores indica que estos valores se han escalado antes de calcular la distancia. Eso es especialmente importante en casos como éste, en que el campo precio tiene valores varios órdenes de magnitud mayores que el campo metros cuadrados.

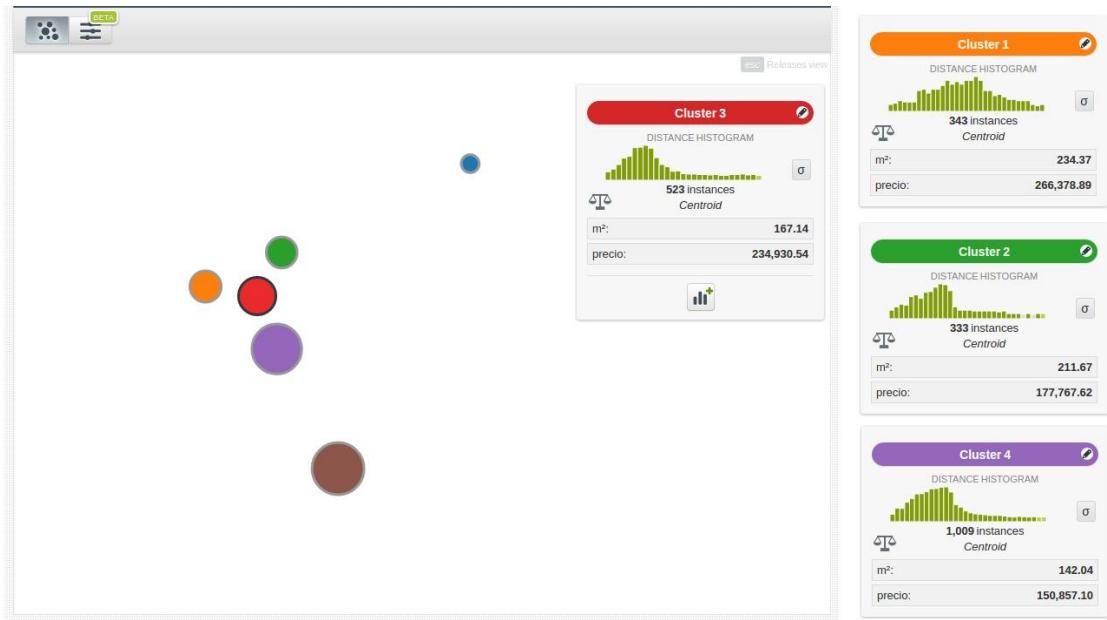


Figura 4.4: Visualización de los *clusters* creados con *g-means*: comparación de las características de los *clusters* más cercanos al central.

Si inspeccionamos otros *clusters* podremos ver qué tipo de instancias contienen y cuáles son las características que las agrupan. Por ejemplo, en el gráfico Figura 4.4 vemos que los *clusters* más cercanos al *cluster* rojo contienen inmuebles con superficies y precios más similares. En cambio, el *cluster* marrón y el azul se diferencian más de ellos. Podemos analizar el detalle de los inmuebles que se han agrupado en el *cluster* azul creando un dataset que contenga esos datos. La Figura 4.5 muestra los histogramas de dicho dataset. Los 88 inmuebles que contiene son bastante más grandes de lo habitual, y su precio por tanto también está en lo más alto del rango.



Figura 4.5: Visualización de los datos agrupados en un *cluster*.

Si queremos una vista más global de cómo se agrupan los datos podemos crear un *Batch centroid*. Esta acción está disponible en el menú de nuestro *cluster*. Seleccionando el *Dataset* que usamos originalmente para crear el *cluster*, el *Batch centroid* asigna el nombre del *cluster* al que pertenece cada fila. El resultado puede ser descargado en formato CSV o podemos generar un nuevo *Dataset* con una columna añadida que contendrá esta etiqueta. La Figura 4.6 nos muestra el resultado de crear un nuevo *Dataset* con un *Batch centroid*.

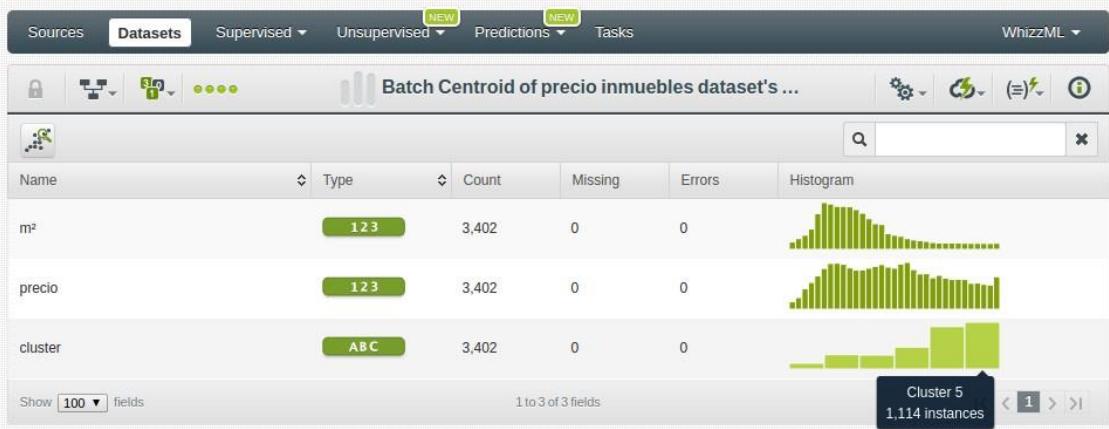


Figura 4.6: Dataset generado a partir de un *Batch centroid*. El *Batch centroid* asigna el nombre del *cluster* al que pertenece cada una de las filas del Dataset original.

Para visualizar mejor esta información, podemos usar el gráfico llamado *Dynamic scatterplot* usando el botón que se muestra en la Figura 4.7.

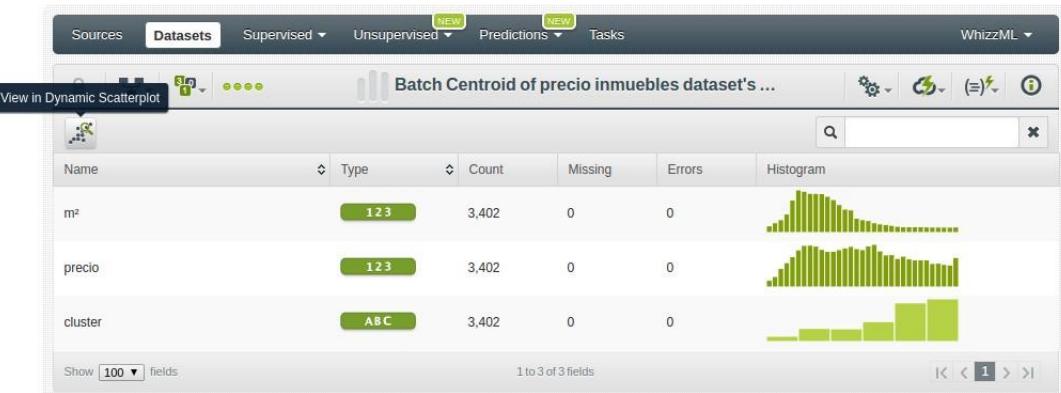


Figura 4.7: Acción que genera un *Dynamic scatterplot* desde un Dataset.

En él se representa un subconjunto de los puntos del Dataset elegidos al azar en un gráfico cartesiano de dos ejes. Podemos elegir qué campo queremos representar en cada eje y también podemos elegir otro campo para que los colores se asignen según su contenido. En nuestro caso, la Figura 4.8 usa los metros cuadrados y el precio como ejes y los colores se han asignado según el nombre del *cluster*. Se observa claramente las regiones definidas por los distintos *clusters* y cuál es la distribución de datos en esas regiones.

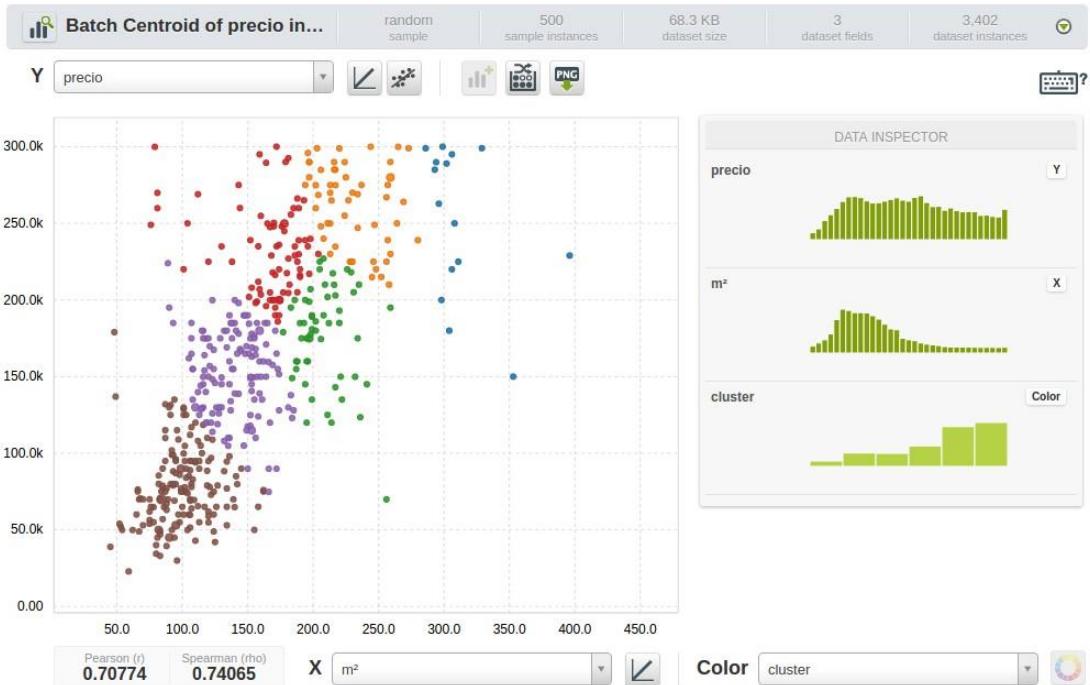


Figura 4.8: *Dynamic scatterplot* del *Dataset* generado por un *Batch centroid*. Los datos muestran un subconjunto de las filas de nuestro dataset coloreadas según el *Cluster* en qué han sido clasificadas.

FIJANDO EL NÚMERO DE GRUPOS: *K-MEANS*

Si sabemos el número de grupos que queremos obtener, podemos dar esa información al crear el *clustering*. Ese número de clusters que queremos obtener será la *k* en nuestro algoritmo *k-means*.

En *k-means* cada fila de nuestro fichero corresponde a un punto en un espacio tantas dimensiones como atributos nuestro dataset. Inicialmente, el algoritmo elige *k* puntos al azar dentro de este espacio como puntos de referencia. Luego elige la primera instancia de nuestro dataset y calcula la distancia de ésta a cada uno de ellos. Finalmente, vincula esa instancia al punto de referencia más cercano. Sucesivamente, se repite el cálculo para todas las instancias del dataset. Con este proceso se consigue formar *k* grupos con las instancias del dataset según su cercanía a los tres puntos iniciales, pero todavía falta conseguir que estos grupos sean los más compactos entre sí y distintos de los demás.

Para ello, se calcula el punto central de todas las instancias pertenecientes a cada grupo. A partir de aquí, se repetirá el proceso anterior, pero eligiendo como nuevos puntos de referencia los tres centros, con lo que se calcularán las distancias de cada instancia a cada uno de los centros y se vincularán con el más cercano. Generalmente, este proceso hará que las instancias se vayan recolocando, formando nuevos grupos que tendrán nuevos centros. Cuando los grupos generados se mantengan estables (las instancias se mantengan en el mismo grupo durante varias iteraciones) habremos terminado el proceso y dichos grupos serán los clusters generados.

Para poner un ejemplo, usaremos un *Dataset* como el que vemos en la Figura 4.9, que contiene las puntuaciones que algunos usuarios han otorgado a un listado de marcas de cerveza.

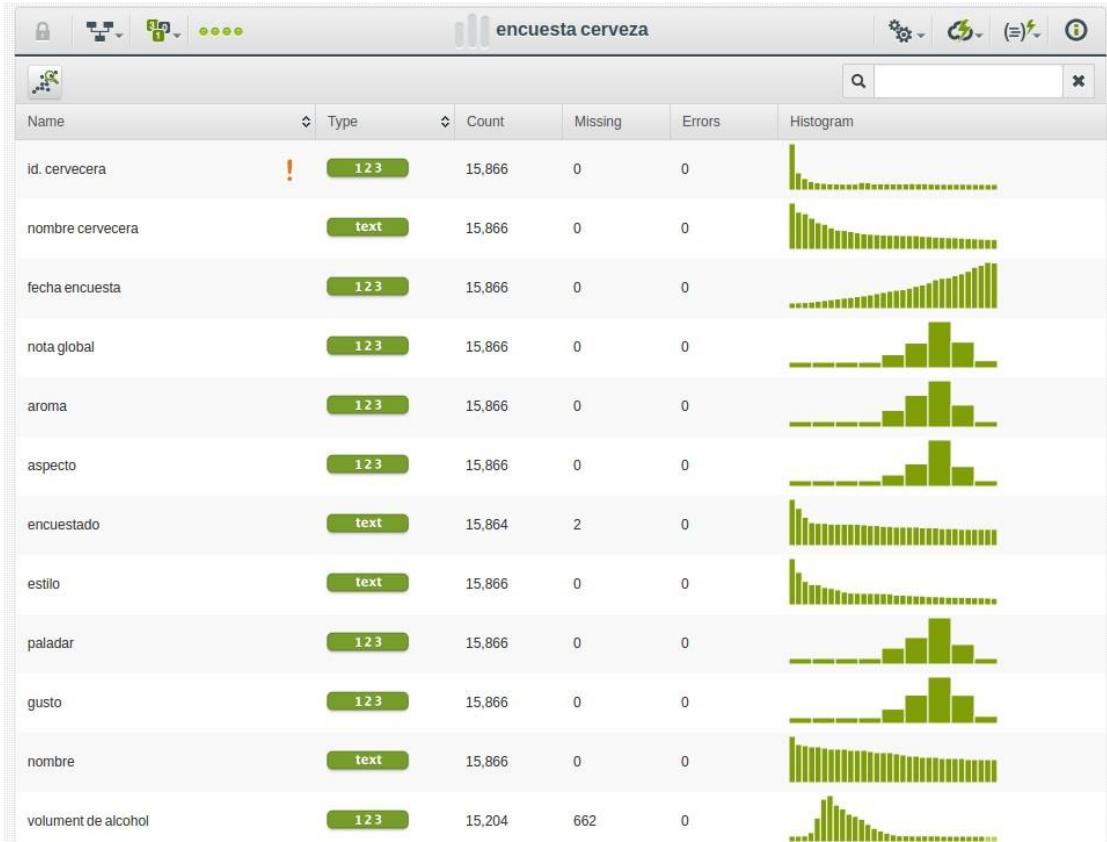


Figura 4.9: Dataset de cervezas: Cada instancia contiene las puntuaciones otorgadas por un usuario a una marca de cerveza según su aroma, paladar, aspecto, gusto así como el volumen de alcohol que contiene.

Nuestro objetivo es clasificar dichas marcas en tres grupos para poder recomendar alternativas a nuestros clientes. Sabiendo el grupo al que pertenecen las cervezas que compran normalmente, podemos ofrecer otras marcas de características similares. Los datos con que contamos incluyen algunos datos que se usan como información de referencia, como el nombre de la cerveza, el de la compañía cervecera, o el identificador del usuario que puntúa. Los atributos que realmente califican la cerveza son la nota global, su aroma, gusto, aspecto, paladar y el volumen de alcohol. Este tipo de atributos son los que nos dicen si una cerveza es similar a otra o no y por lo tanto serán los que nos interesarán usar en el clustering.



Figura 4.10: Configuración personalizada de un cluster.

Construiremos pues un modelo de *clustering* usando la pantalla de configuración de *Cluster*, tal como vemos en la Figura 4.10. Además de fijar el número de clusters a generar ($k=3$), también marcaremos el botón de modelado y listaremos los campos que hemos mencionado en el apartado anterior como campos de referencia. Estas opciones se muestran en la Figura 4.11, pero vamos a explicarlas un poco más para entender qué resultado perseguimos con ellas.

The screenshot shows the WhizzML interface with the 'Datasets' tab selected. The dataset 'encuesta cerveza' is loaded. In the 'CLUSTER CONFIGURATION' section, the 'Clustering algorithm' is set to 'K-means', 'Number of clusters (K)' is set to 3, and the 'Model clusters' button is checked. Under 'Advanced configuration', the 'Scales' and 'AUTOSCALED FIELDS' sections are set to 'No'. The 'Weights' and 'Summary fields' sections list various fields: 'encuestado', 'estilo', 'nombre cervecera', 'fecha encuesta', and 'nombre'. In the 'Sampling' section, it shows '15,866 instances'. At the bottom, there is a 'Create cluster' button.

Figura 4.11: Pantalla de configuración de un *Cluster*: las opciones usadas son $k=3$, botón de modelado activado y un listado de campos de referencia (*summary fields*).

Los campos listados bajo el epígrafe *summary fields* son tenidos en cuenta solamente a nivel informativo, pero nunca son usados en el cálculo de la similitud entre los datos. Vimos ya, en el apartado anterior, que se puede crear un *Dataset* que contenga solamente los datos agrupados en un mismo *cluster*. Seguramente, en ese *Dataset* nos interesaría tener la información de referencia, como el nombre de la cerveza analizada, y sin embargo no querremos que la diferencia de nombre sea tenida en cuenta cuando analizamos si las cervezas se parecen o no. Por eso, incluiremos el campo que contiene el nombre de la cerveza en los *summary fields* y, como éste, cualquier otro campo que contenga informaciones que queramos conservar pero que no influyen en la similitud de nuestras instancias.

El otro cambio de configuración es el botón de modelado. Al activarlo, se generará un árbol de decisión para cada *cluster*. El objetivo de dicho árbol de decisión es predecir si una instancia pertenece o no al *cluster* en cuestión. El modelo se construye usando el dataset original del que partimos con una columna más, donde se ha añadido la información sobre si la instancia pertenece al cluster o no.

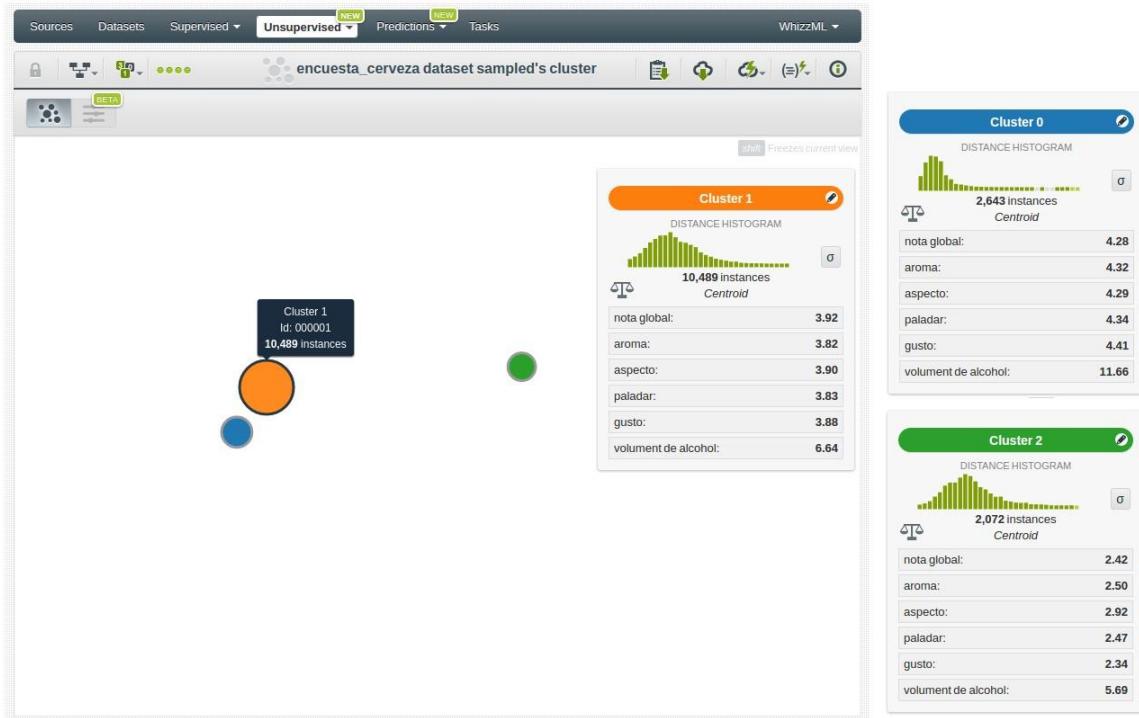


Figura 4.12: Visualización del *Cluster* para el *Dataset* de la encuesta sobre cervezas. Los *Centroids* presentan el promedio de las características de cada grupo.

Los *centroids* de los *clusters* obtenidos con este nuevo algoritmo se muestran en la Figura 4.12. Podemos ver un resumen de las distancias entre los puntos de un mismo *cluster* y también entre los distintos *centroids* usando el botón del *Cluster Summary Report*, tal como se muestra en Figura 4.13. Eso nos da una idea de cómo de compactos son los grupos y cuan distintos son entre sí.

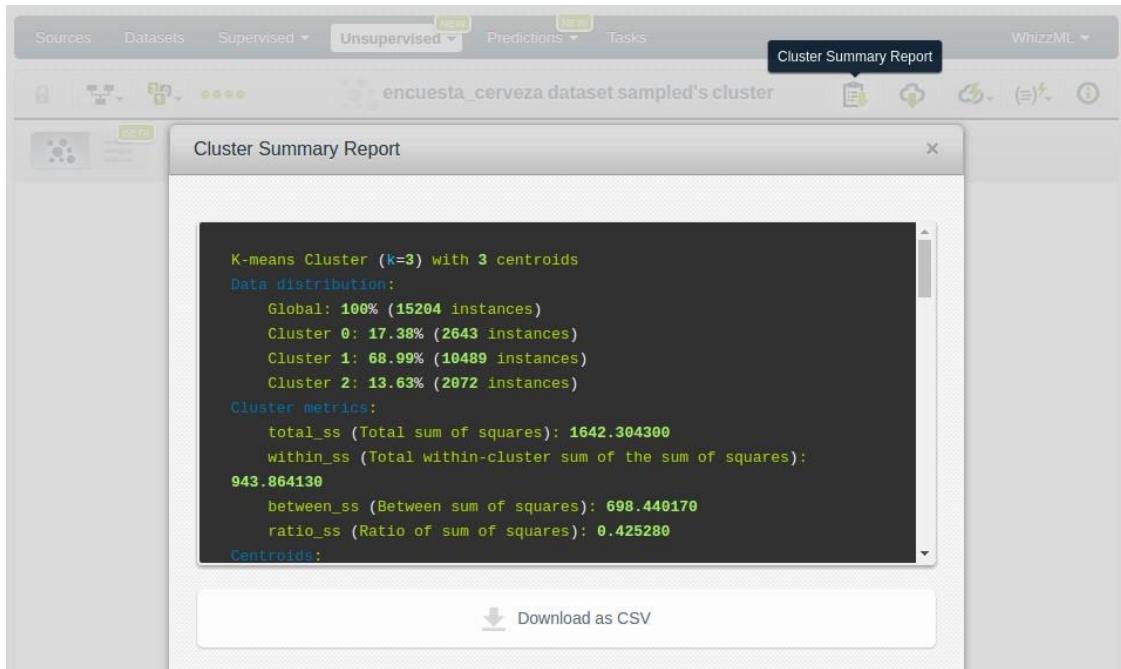


Figura 4.13: Resumen de las distancias entre los puntos pertenecientes a cada *cluster* y las distancias entre los *centroids*.

Aparentemente, la diferencia más evidente entre ellos es el volumen de alcohol, pero para entender qué factores hacen realmente que una instancia pertenezca a un cluster concreto podemos usar los modelos asociados que hemos creado al activar el botón correspondiente en la configuración del cluster (Figura 4.11). Para obtener el modelo correspondiente al *cluster 0*, deberemos apretar la tecla *mayúsculas* al pasar el ratón sobre él. Se mostrará el botón que vemos en la Figura 4.14, que nos permitirá acceder al modelo que predice la pertenencia a ese cluster.

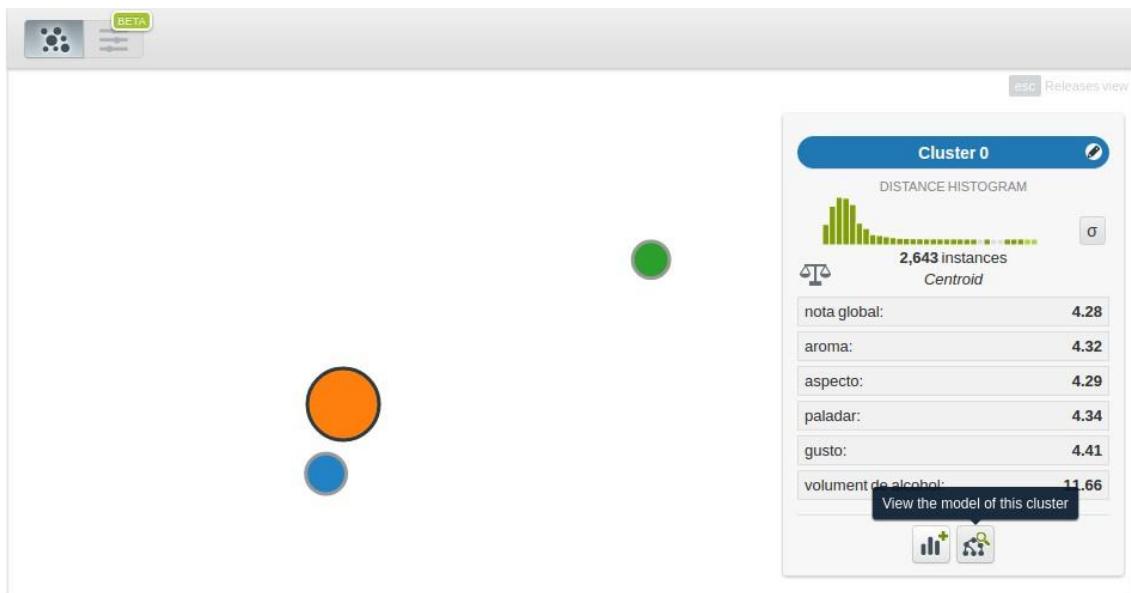


Figura 4.14: Botón de creación de modelos: Pulsando el botón crearemos el modelo que predice la pertenencia o no al *cluster 0*.

Si miramos la importancia de cada campo en ese modelo (Figura 4.16) vemos que el volumen de alcohol es precisamente el campo que más influye, seguido por el gusto y el paladar.

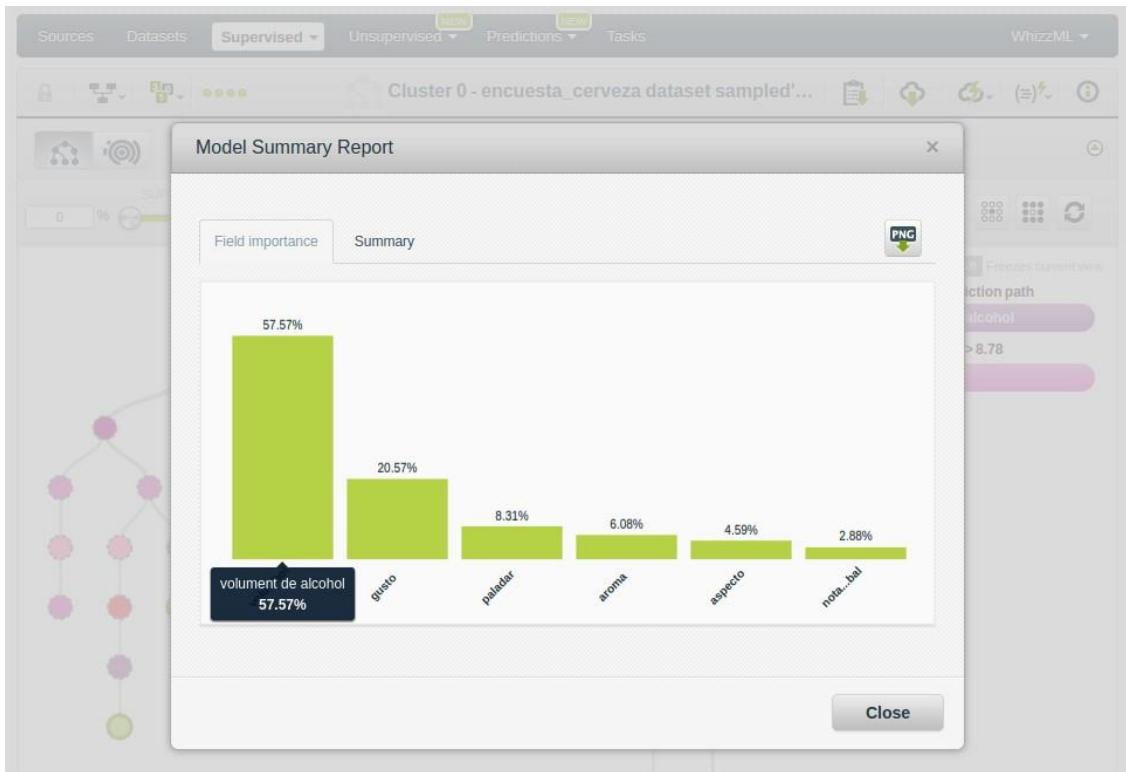


Figura 4.15: Importancia de los campos en el modelo de pertenencia al *cluster 0*.

Repetiendo el proceso que vimos en el ejemplo del dataset de inmuebles con que ilustramos el algoritmo de *g-means*, si creamos un dataset que almacene qué *cluster* se asigna a cada instancia y vemos su *scatterplot* eligiendo estos campos como ejes del gráfico, nos será fácil entender la distribución de las cervezas en cada grupo según sus propiedades (Figura 4.17). Según vemos, las cervezas más suaves tanto por gusto como por volumen de alcohol se agrupan en un *cluster*, mientras que las que tienen más gusto se separan en dos dependiendo de si su volumen de alcohol es alto o bajo.



Figura 4.16: Importancia de los campos en el modelo de pertenencia al *cluster 0*.

Con estas informaciones, podemos escoger alternativas a las marcas habituales de nuestros clientes buscando primero el *cluster* en que se clasifican y luego las otras marcas con características más parecidas dentro de este grupo.



Figura 4.17: Scatterplot que representa la distribución de los tres *clusters* descubiertos en el *Dataset* de la encuesta sobre cervezas en función de su gusto y el volúmen de alcohol.

4.2. Detección de anomalías: Anomaly Detector

En los problemas de detección de anomalías el objetivo es encontrar aquellas instancias que se distinguen porque no siguen los patrones presentes en el resto de los datos. Para conseguir aislar estas instancias del resto, utilizaremos un detector de anomalías basado en un algoritmo conocido como *isolation forest*.

El algoritmo se basa en la idea de que, si una instancia es muy anómala, como algunos de sus atributos serán muy distintos de lo habitual será fácil caracterizarla y separarla. Para hacerlo, el algoritmo construye un conjunto de árboles de decisión. Cada uno de ellos parte de un subconjunto distinto de los datos del *Dataset* escogido al azar. El objetivo de estos árboles es aislar cada instancia en un nodo distinto a base de establecer condiciones sucesivas sobre sus atributos. Estas condiciones van separando los grupos de instancias a medida que se baja en profundidad por el árbol. Las instancias que quedan a poca profundidad en el árbol se considera que son más anómalas, ya que se han podido diferenciar de las demás usando pocas condiciones. En cambio, las que quedan a mucha profundidad serán menos anómalas, ya que estas han requerido muchas más condiciones para diferenciarse de las demás al ser más parecidas. Basándose en esta observación, podemos asociar un *anomaly score* a cada instancia, comparando la profundidad a la que se separa en cada árbol con la profundidad media de los árboles construidos, tal como se muestra en la Figura 4.18.

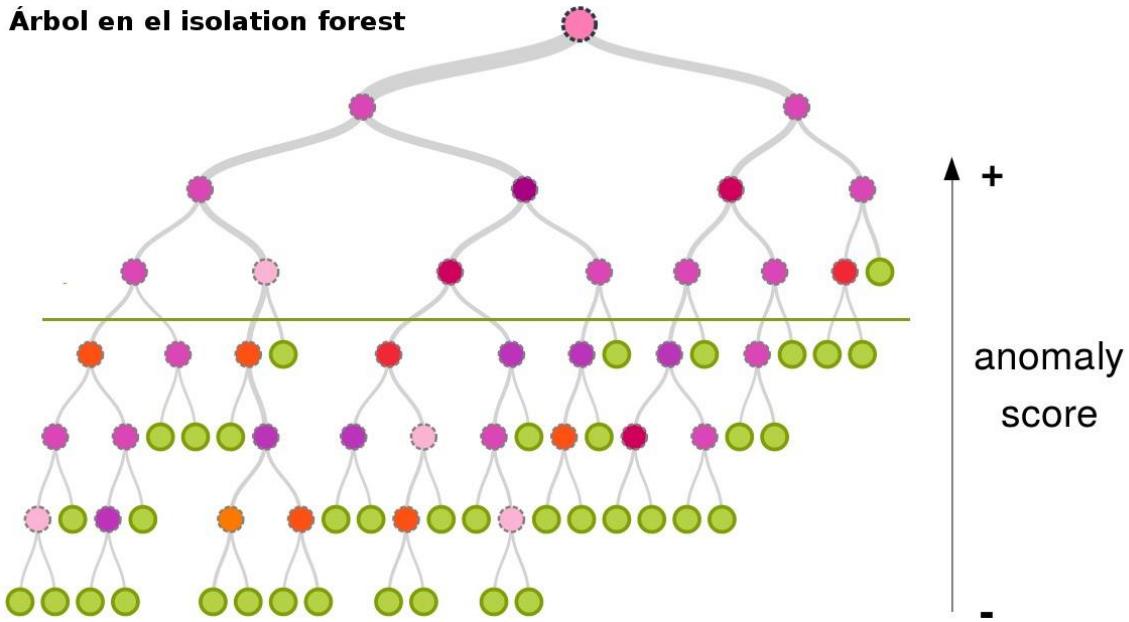


Figura 4.18: Esquema del cálculo del *anomaly score*. La línea horizontal marca la profundidad media de los árboles generados y las instancias se consideran más anómalas cuanto más arriba están los nodos en que se separan del resto.

El *anomaly score* es, por consiguiente, un número entre 0 y 1 que caracteriza el nivel de anomalía de cada instancia. El valor 0 indica total ausencia de anomalías y el 1 la máxima anomalía posible. En general, consideraremos que las instancia con un *anomaly score* mayor que 0,6 son anómalas, pero este límite puede variar según la distribución de nuestros datos.

Para ver cómo crear un detector de anomalías y qué informaciones podemos obtener con un detector de anomalías, usaremos un dataset donde tenemos algunos datos sobre préstamos. La información de que disponemos se muestra en la Figura 4.19 e incluye, entre otras variables, la cuantía del préstamo, los intereses, si hay impagos y la calificación del prestatario.

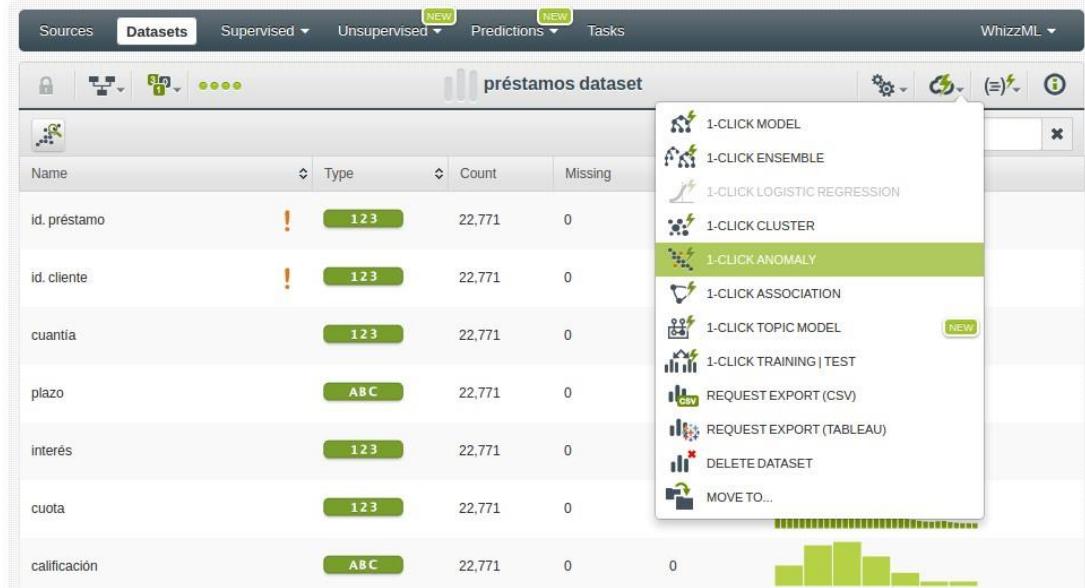


Figura 4.19: Creación de un detector de anomalías. Podemos crear un detector de anomalías en un clic desde la pantalla del *Dataset*. En el ejemplo, los atributos incluyen variables como la cuantía del préstamo, los intereses, los posibles impagos y la calificación del prestatario.

Partiendo de este *Dataset*, podemos crear un detector de anomalías mediante el enlace *1-click Anomaly*. De esa forma, se usarán los valores por defecto de los parámetros de configuración, como el número de árboles de decisión usados en el algoritmo y las n instancias más anómalas que se mostarán (por defecto las diez primeras). El resultado de esta acción se muestra en la Figura 4.20. A la izquierda de la pantalla encontramos la lista de instancias más anómalas.

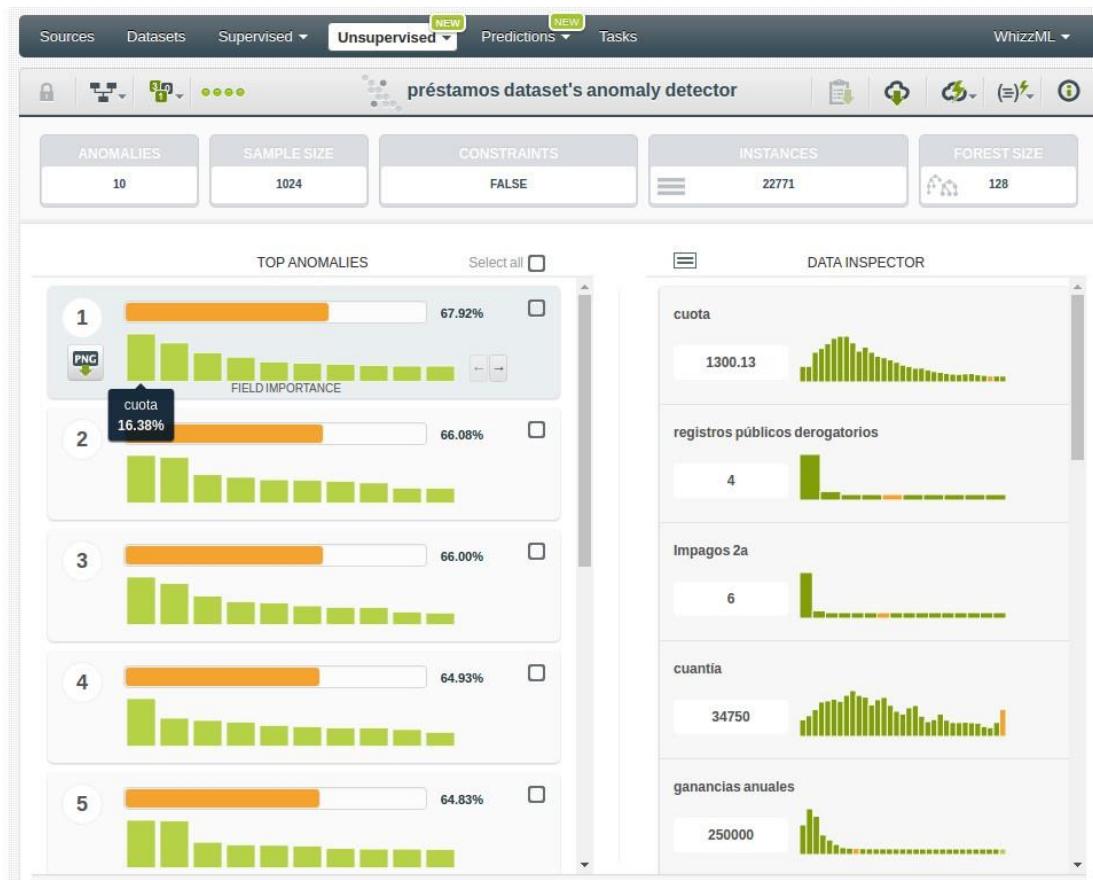


Figura 4.20: Visualización del detector de anomalías. A la izquierda se listan las 10 instancias más anómalas. A la derecha se muestran los campos que contribuyen a la anomalía de la instancia seleccionada y sus valores.

Fíjemonos en la primera instancia de la lista. Su *anomaly score* es de 0,6792 y el gráfico de barras muestra la importancia que cada campo tiene en dicha anomalía. Cuando se selecciona una instancia con el ratón, se muestran a su derecha los valores de sus atributos. También se marca en naranja dónde caen esos valores dentro del histograma general formado por todos los valores del campo. Por ejemplo, vemos que el campo que contribuye más a marcar la primera instancia como anómala es la *cuota*. Su cuantía es de las más altas que se pueden encontrar en nuestros datos. Por eso la barra naranja que muestra su ubicación en el histograma del campo está muy a la derecha, coincidiendo con los valores más altos.

Si miramos el resto de valores de los campos de esta instancia, vemos que se trata de un préstamo con una cuota bastante alta, varios impagos en los últimos años, un importe alto y una calificación E, que es una de las

peores. Este préstamo es altamente irregular, ya que el prestatario no cumple con las condiciones usuales para poder conseguir préstamos de esas características. Las demás anomalías detectadas también presentan valores en sus campos que las hace inusuales desde el punto de vista de un experto financiero. No obstante, hay que recordar que el detector de anomalías las ha encontrado simplemente comparando sus valores con los más habituales y sin tener ningún conocimiento previo sobre las reglas usuales de los productos financieros. Por lo tanto, el modelo nos servirá para cualquier conjunto de datos, sin importar su naturaleza ni el dominio en el que se usen.

Tras detectar las anomalías de un *Dataset* tenemos varias opciones. Podemos crear un nuevo *Dataset* con ellas, para poder analizarlas mejor, o podemos extraerlas del *Dataset* original, para *limpiarlo* de casos anómalos. A menudo, esta eliminación de las instancias más anómalas de nuestro *Dataset* permite generar mejores modelos predictivos con los datos restantes. Para ello, podemos seleccionar las anomalías a incluir o excluir y usar el botón que hay al final de la pantalla para fijar la opción elegida al crear el nuevo *Dataset*.



Figura 4.21: Creación de un nuevo *Dataset* donde se incluyen o excluyen las anomalías seleccionadas. En este caso, se generaría un *Dataset* con una única instancia seleccionada: la más anómala. Si, por el contrario, queremos excluirla del *Dataset* original, deberemos marcar el botón resaltado en la imagen.

Como hemos mencionado, el detector de anomalías construido por defecto muestra las diez instancias con mayor *anomaly score* encontradas en nuestro *Dataset*. No obstante, puede ocurrir que haya más o menos instancias anómalas. Para tener una visión global del cómo se distribuye el *anomaly score* en nuestros datos podemos crear un *batch anomaly score*, tal como se muestra en la Figura 4.22.



Figura 4.22: Creación de un *Batch anomaly score*: Se asignará un *anomaly score* a cada instancia usando el detector de anomalías.

Con este proceso, podremos obtener un nuevo *Dataset* donde, además de los datos originales, aparecerá una columna adicional que contendrá el *anomaly score* asignado por el detector de anomalías a cada fila. Analizando la distribución de valores de este nuevo campo, podremos saber si nuestros datos son más o menos homogéneos y, en función de eso, situar el valor del *anomaly score* que mejor separa las instancias anómalas. En la Figura 4.23 se muestra la distribución para nuestro ejemplo. Por la forma de la distribución, vemos que en este caso el valor límite de 0,6 es bastante ajustado, porque es donde empieza la cola de la gausiana.



Figura 4.23: *Batch anomaly score*: el *Dataset* generado mediante un *Batch anomaly score* contiene una última columna que almacena el *anomaly score* de cada una de las filas del *Dataset* original. La imagen muestra la distribución de sus valores.

Gracias a este histograma, podremos fijar el límite de *anomaly score* que nos parezca adecuado y filtrar los datos en función de él utilizando los métodos de filtrado de *Datasets* vistos en el Módulo 2.

4.3. Reglas de asociación: Association Rules

El último tipo de problema que nos plantearemos será el de encontrar posibles asociaciones entre valores de distintos campos que se dan a la vez con más frecuencia de la que cabría esperar. Estas asociaciones no son correlaciones, es decir, no buscamos campos que estén relacionados para todos sus valores. Tampoco es exactamente un problema de causalidad. Se buscan coocurrencias, es decir, diremos que hay una regla de asociación (*association rule*) cuando un predicado sobre un campo (por ejemplo, *campo1 = valor1*) ocurre a menudo a la vez que otro predicado (por ejemplo, *campo2 > valor2*). El ejemplo típico de este tipo de problema es el análisis de la cesta de la compra, donde se persigue saber qué productos se adquieren juntos para poder adecuar ofertas o su disposición en los expositores.

Hay que tener en cuenta que para que realmente haya una asociación relevante, la frecuencia con que coinciden ambos valores ha de ser mayor que la que se daría si se eligieran los dos independientemente por puro azar. Por eso, será necesario decidir cómo determinaremos si la coocurrencia observada es realmente significativa.

Usaremos un fichero de compras para ilustrar este tipo de modelos. El objetivo es pues encontrar relaciones como: *si se compra panecillos y hamburguesas, normalmente también se compra ketchup*. Como vemos, el tipo de reglas que expresarán estas asociaciones tienen el formato:

Antecedente ⇒ Consecuente

En el ejemplo, el antecedente sería *compra = panecillos y compra = hamburguesas* y el consecuente *compra = ketchup*. El antecedente puede contener más de un valor, mientras que en el consecuente sólo tendremos uno. Otros problemas en los que puede ser útil la detección de asociaciones son en los casos de análisis de fallos, detección de intrusos y biotecnología.

De entrada, vamos a analizar qué aspecto tendrán los datos que se usarán en este tipo de análisis. Al tratarse de una cesta de la compra, cada fila contendrá la información correspondiente a un tique de caja y en las columnas habrá el detalle del tipo de productos adquiridos. La estructura será pues similar a la de la [Figura 4.24](#).

	A	B	C	D	E	F	G	H	I
1	cítricos	pan precocinado	margarina	sopas instantáneas					
2	fruta tropical	yogurt	café						
3	leche entera								
4	fruta de pepita	yogurt	crema de queso	paté de carne					
5	otros vegetales	leche entera	leche condensada	galletas					
6	leche entera	mantequilla	yogurt	arroz	limpiador abrasivo				
7	bollos								
8	otros vegetales	leche pasteurizada	bollos	cerveza embotellada	vermut				
9	macetas								
10	leche entera	cereales							
11	fruta tropical	otros vegetales	pan blanco	agua embotellada	chocolate				
12	cítricos	fruta tropical	leche entera	mantequilla	cuajada	yogurt	harina	agua embotellada	platos
13	ternera								
14	frankfurt	bollos	soda						
15	pollo	fruta tropical							
16	mantequilla	azúcar	fruta/zumos	periódicos					
17	fruta/zumos								
18	fruta/vegetales empaquetados								
19	chocolate								

[Figura 4.24](#): Estructura de los datos usados en el análisis de la cesta de la compra. Cada fila contiene información sobre un tique de caja. El número de columnas es variable y contiene los tipos de producto comprados.

Hay otros formatos alternativos igualmente válidos para el análisis. Podemos tener columnas con información de referencia, como el número de tique o la fecha de compra y todos los productos adquiridos acumulados en una sola columna y separados por un carácter distintivo. La [Figura 4.25](#) muestra un ejemplo de este tipo de estructura.

	A	B
1	tique	productos
2	342533	cítricos,pan precocinado,margarina,sopas instantáneas
3	342534	fruta tropical,yogurt,café
4	342535	leche entera
5	342536	fruta de pepita,yogurt,crema de queso,paté de carne
6	342537	otros vegetales,leche entera,leche condensada,galletas
7	342538	leche entera,mantequilla,yogurt,arroz,limpiador abrasivo
8	342539	bollos
9	342540	otros vegetales,leche pasteurizada,bollos,cerveza embotellada,vermut
10	342541	macetas
11	342542	leche entera,cereales
12	342543	fruta tropical,otros vegetales,pan blanco,agua embotellada,chocolate
13	342544	cítricos,fruta tropical,leche entera,mantequilla,cuajada,yogurt,harina,agua embotellada,platos
14	342545	ternera
15	342546	frankfurt,bollos,soda
16	342547	pollo,fruta tropical
17	342548	mantequilla,azúcar,fruta/zumos,periódicos

Figura 4.25: Estructura de los datos usados en el análisis de la cesta de la compra. Cada fila contiene informaciones de referencia, como el número de tique o la fecha de compra, y un único campo contiene los tipos de producto comprados, que se han separado mediante un carácter distintivo.

Al subir los datos de la [Figura 4.25](#), el objeto *Source* interpreta la columna que contiene los productos como un campo de tipo *Items*, con lo que analizará cada uno de los valores separados por comas como una unidad. Al crear el *Dataset* correspondiente podremos ver las frecuencias con que aparece cada uno de ellos.

Volviendo al ejemplo original de la Figura 4.24, el *Dataset* correspondiente muestra que el producto comprado con más frecuencia es la *leche entera*, seguido de *otros vegetales*, *bollos*, etc. El histograma correspondiente se muestra en la Figura 4.26 y nos ofrece la información estadística de la distribución de los productos en todas las compras, pero usando Machine Learning seremos capaces de descubrir cuáles se compran a la vez frecuentemente.



Figura 4.26: Dataset generado a partir del fichero de cestas de la compra. La nube de términos muestra los tipos de producto. El tamaño de letra es proporcional a la frecuencia con que se da cada término.

A partir del *Dataset*, podremos crear un detector de asociaciones usando la acción *1-click Association* tal como vemos en la [Figura 4.27](#).



Figura 4.27: Creación de un detector de asociaciones en un clic desde un Dataset.

Como resultado, obtenemos una lista de 100 reglas que podemos ver en la Figura 4.28. La primera regla nos dice que cuando se compran *tubérculos* también se compran *otros vegetales*.

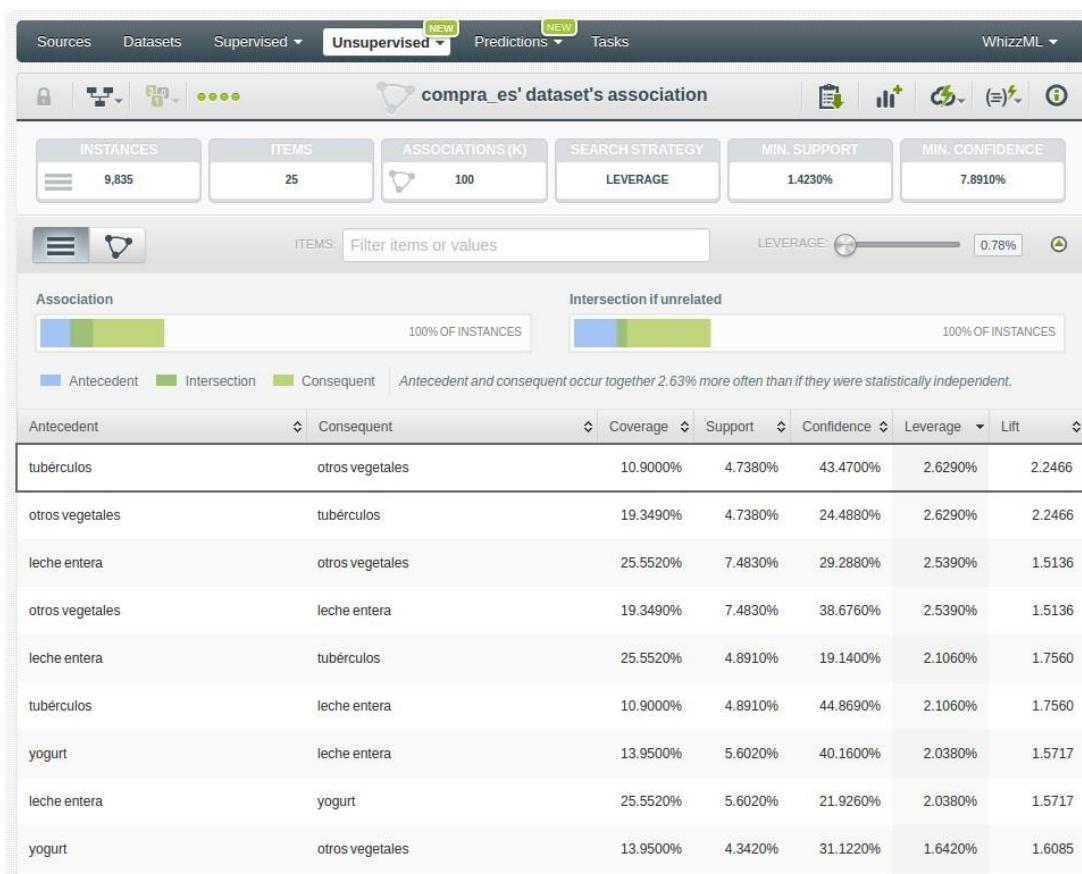


Figura 4.28: Reglas de asociación. La pantalla muestra las cien reglas de asociación que mejor apalancamiento proporcionan.

En la parte superior de la imagen vemos unas barras de colores azul y verde. Tal como explican las leyendas, la parte coloreada en azul representa las instancias que tienen el antecedente, en este caso las cestas donde se han comprado *tubérculos*. La parte coloreada en verde claro representa las instancias del consecuente, en el ejemplo son las cestas donde se han comprado *otros vegetales*. Vemos que hay una intersección coloreada en un verde azulado, que representa los casos en que ambos productos coinciden en una cesta. En el gráfico de la derecha la intersección corresponde a las cestas que contendrían ambos productos si éstas se creasen eligiendo productos al azar de entre todos los existentes. En cambio, en el gráfico de la izquierda la intersección muestra el número real de cestas que contienen ambos productos en la muestra de nuestros datos. Como podemos ver,

el número de cestas que los contienen es mucho mayor que el que se daría por una elección al azar, por lo tanto, podemos establecer que la compra de ambos productos está relacionada.

Hemos podido, pues, establecer que existe una asociación entre dos productos, pero sería interesante poder medir de algún modo en cuánto supera la intersección producida por la asociación a la que habría si esa asociación no existiese. Existen varias métricas que por si solas o combinadas nos permitirán cuantificar esa intensidad de la relación:

- Cobertura: Es el porcentaje sobre el total de casos en que se da el antecedente.
- Soporte: Es el porcentaje sobre el total de casos en que aparecen tanto el antecedente como el consecuente. Confianza: Es el porcentaje, sobre el total de casos que cumplen el antecedente, donde se da el consecuente.
- Apalancamiento (o Leverage): Es la diferencia entre el soporte existente y el que se daría si antecedente y consecuente fuesen independientes.
- Mejora (o Lift): Es la proporción de casos en que antecedente y consecuente se dan juntos comparados con los que se darían si fuesen independientes.

La [Figura 4.28](#) muestra los valores de dichas métricas para cada una de las reglas de asociación detectadas. Por tanto, sobre la primera regla de nuestro ejemplo podemos decir que:

- La cobertura es del 10,9% porque los *tubérculos* aparecen en ese porcentaje de las compras.
- El soporte es del 4,738% porque los *tubérculos* aparecen junto a los *otros vegetales* en ese porcentaje de cestas.
- La confianza es del 43,47% porque los *otros vegetales* aparecen en ese porcentaje de las cestas que contienen *tubérculos*.
- El apalancamiento es del 2,629% porque los *tubérculos* y los *otros vegetales* aparecen juntos 2,63% más que si no estuviesen relacionados.
- La mejora es del 2,25 porque los *otros vegetales* aparecen 2,25 veces más en las cestas que tienen *tubérculos*.

Dependiendo de cuál sea nuestro objetivo, podemos usar cualquiera de estas métricas como criterio para seleccionar las n reglas de asociación que mejor puntúen en esa métrica. Por defecto, el detector de asociaciones selecciona las reglas que tienen mejor apalancamiento, pero este criterio se puede modificar usando la pantalla de configuración del detector de asociaciones de la [Figura 4.29](#).

Figura 4.29: Pantalla de configuración del detector de asociaciones. Podemos decidir el número máximo de reglas seleccionadas y el criterio que guía su búsqueda se puede escoger entre cobertura, soporte, confianza, apalancamiento o mejora.

Si cambiamos el criterio de búsqueda del detector de asociaciones para usar la *mejora* en vez del *apalancamiento* (que es el criterio usado por defecto) recuperaremos otras reglas, que no ocurrirán tan a menudo en el total de compras, pero en que la vinculación entre antecedente y consecuente puede ser más fuerte. La Figura 4.30 muestra dichas reglas para nuestros datos de ejemplo. En la primera regla vemos como el *licor* aparece 33 veces más cuando la *cerveza embotellada* y el *vino tinto* están también en la cesta, aunque en realidad la cantidad de compras de estos productos es menor.

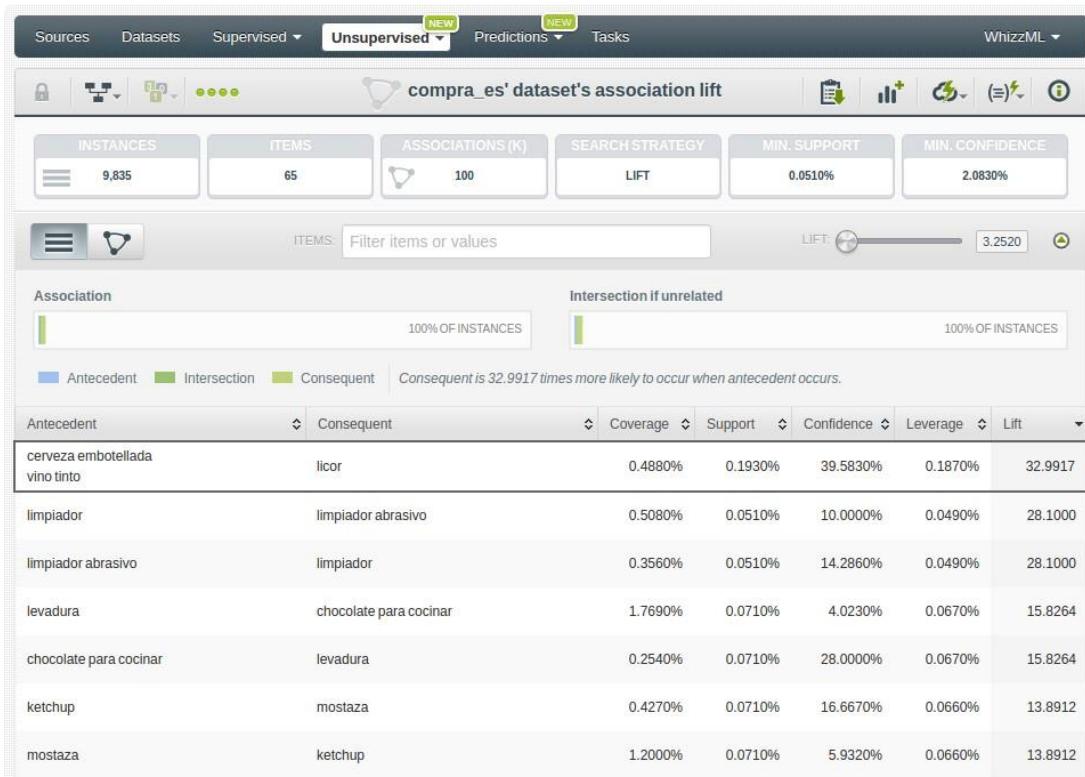


Figura 4.30: Reglas de asociación. Lista de las reglas recuperadas con mayor mejora.

Si queremos encontrar las reglas detectadas que tienen más *apalancamiento* dentro de las que tienen más *mejora*, podemos cambiar la ordenación de la lista de reglas con un clic sobre la cabecera de la columna *apalancamiento*. El resultado se muestra en la Figura 4.31 y la primera regla nos dice que la *nata* y los *frutos rojos* aparecen juntos un 0.66% más de lo que cabría esperar si se comprasen al azar. También podemos observar que las reglas no siempre son simétricas. Por ejemplo, vemos la regla que dice que si encontramos en la cesta de la compra es probable que también encontrremos *verduras congeladas*. En cambio, la regla inversa no aparece porque el hecho de tener *verduras congeladas* en la cesta de la compra no conlleva habitualmente que también se adquiera *pollo*.

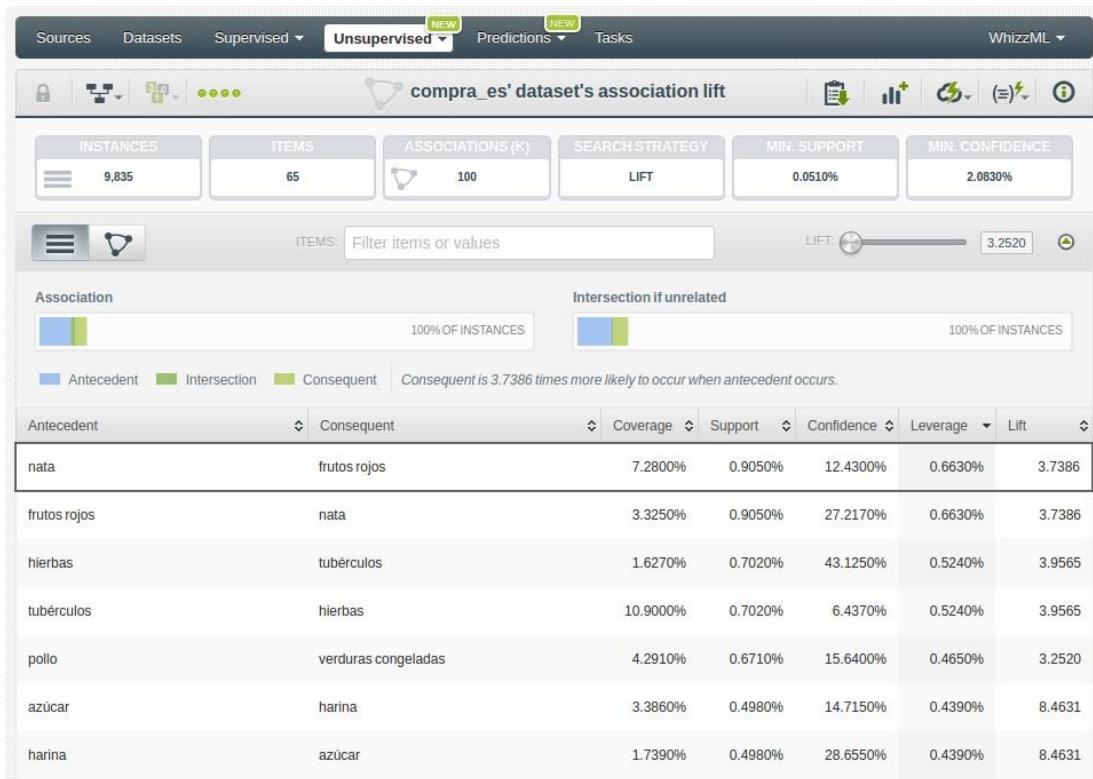


Figura 4.31: Reglas de asociación. Lista de las reglas recuperadas con mayor mejora ordenadas por apalancamiento decreciente.

Además de esta lista exhaustiva de reglas, podemos ver una representación gráfica que nos ofrece una perspectiva global de las relaciones existentes entre los productos usando el botón que se muestra en la Figura 4.32. En este esquema, los círculos representan los productos que forman parte de las reglas. Las curvas que los unen simbolizan las asociaciones que detectadas entre los distintos productos (véase Figura 4.33).

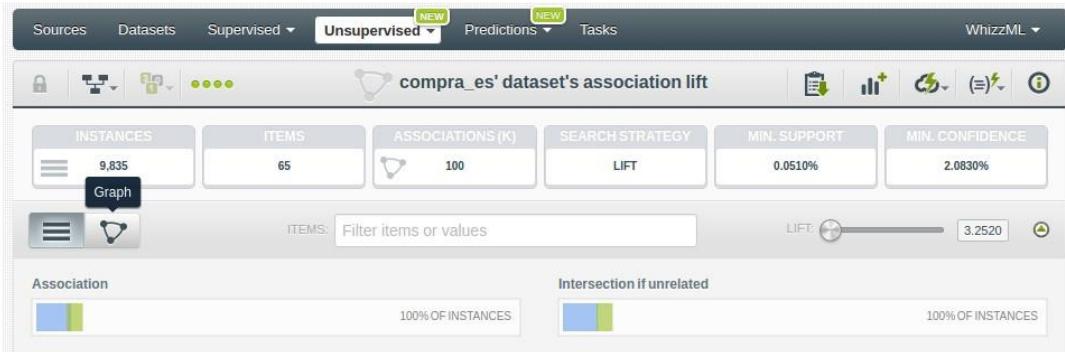


Figura 4.32: Botón de visualización gráfica de las reglas de asociación.

En ambas visualizaciones podemos filtrar las reglas, de manera que se muestren solo las que estén por encima de un cierto nivel de *mejora* (o el criterio que se haya elegido en la búsqueda en su lugar). Al elevar el umbral de *mejora* solo se mostrarán las asociaciones más fuertes. En el gráfico, el grosor de las líneas es un indicador de la magnitud de la *mejora* de la regla en cuestión. Para nuestro ejemplo, la regla con más *mejora* es la que relaciona la *cerveza embotellada*, el *vino tinto* y los *licores*, como ya vimos en el listado.

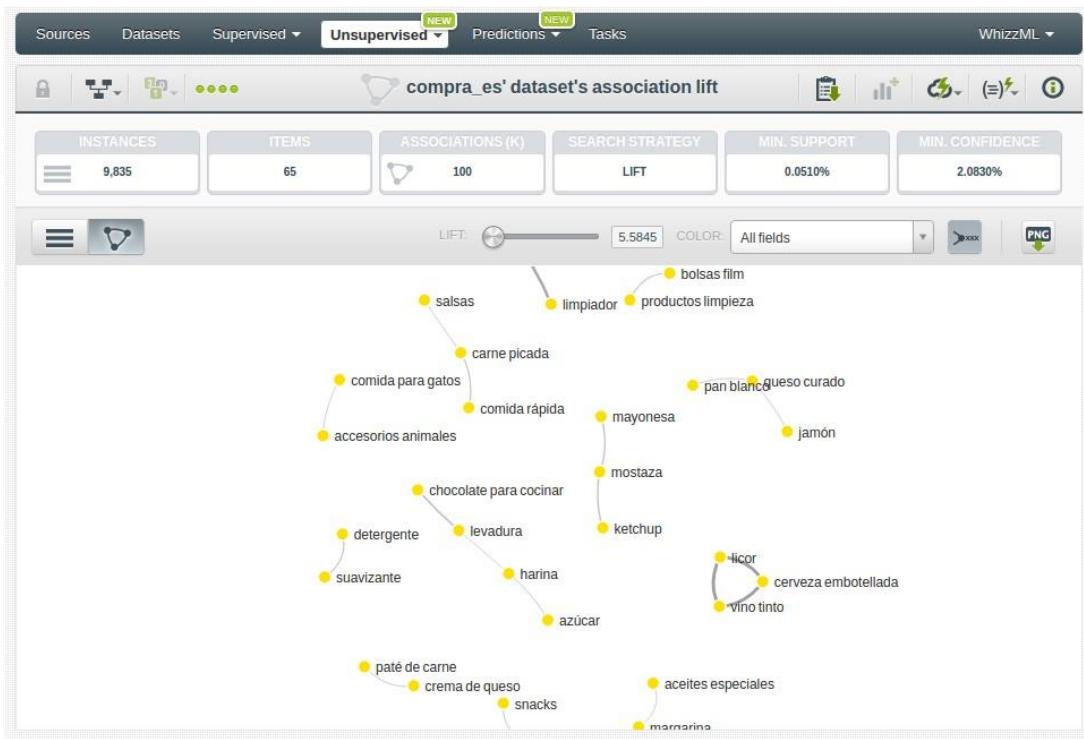


Figura 4.33: Visualización de las reglas de asociación en forma de grafo. Los círculos corresponden a los antecedentes y consecuentes y las líneas que los unen representan las asociaciones entre ellos.

En función de los resultados que queramos obtener, usaremos las reglas con más *apalancamiento o mejora*. Si usamos las primeras, estaremos detectando reglas que se dan con mayor frecuencia, mientras que si usamos las segundas detectaremos asociaciones más fuertes. Podemos igualmente usar cualquiera de las otras métricas en nuestra búsqueda de asociaciones, pero éstas dos son las más recomendadas, ya que normalmente caracterizan las asociaciones más relevantes.