

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR



PROYECTO FIN DE CARRERA

PREDICCIÓN DE FRAUDE EN UN OPERADOR MÓVIL VIRTUAL MEDIANTE INTEGRACIÓN DE DATOS Y DATA MINING

Ingeniería de Telecomunicación

Sergio Izquierdo del Álamo
Noviembre de 2015

PREDICCIÓN DE FRAUDE EN UN OPERADOR MÓVIL VIRTUAL MEDIANTE INTEGRACIÓN DE DATOS Y DATA MINING

AUTOR: Sergio Izquierdo del Álamo
TUTOR: Estrella Pulido Cañabate

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Noviembre de 2015

Resumen

República Móvil es un Operador Móvil Virtual fundado en 2013. La utilización de diversos sistemas operacionales hace necesaria la integración y la consolidación de datos entre éstos para obtener una visión global con suficiente precisión como para realizar análisis avanzado de Inteligencia de Negocio. Para lograr este objetivo, se ha creado un *data warehouse* basado en un almacén de datos normalizado y un almacén de datos dimensional, que puede ser empleado para construir *data marts* departamentales o explotado directamente mediante un servidor de BI.

Utilizando la infraestructura desarrollada, se ha implementado un *data mart* con las medidas y dimensiones necesarias para analizar el riesgo de impago de un cliente en el momento de su alta. Esta información se preprocesa y analiza con Máquinas de Soporte Vectorial (SVMs), tratando de predecir durante el proceso de alta si un cliente va a cumplir con el pago de sus facturas. Por último, se ha conseguido mejorar la capacidad predictiva del modelo empleando Reglas Asociativas para rellenar valores desconocidos en el conjunto de datos.

Palabras Clave

OMV, Operador Móvil Virtual, data warehouse, data mining, data mart, máquinas soporte vectorial, reglas asociativas, pentaho kettle, pentaho data integration, pentaho data mining, pentaho weka, pentaho bi server, impago, riesgo

Abstract

República Móvil is a Madrid based MVNO established in 2013. As multiple operational systems are used, it is necessary to integrate and consolidate the data across them to obtain an enterprise-wide vision that enables performing advanced Business Intelligence analyses. To make this possible, a data warehouse has been deployed. The architecture encompasses a Normalized Data Store and a Dimensional Data Store. The latter can feed departmental data marts, or be analyzed directly using a BI server.

A data mart containing all the necessary dimensions and measures to analyze nonpayment risk of a given client during their sign-up process has been deployed using the implemented infrastructure. In order to predict the future payment behavior of a new client, this information has been preprocessed and mined using Support Vector Machines. Last, the predictive capabilities of the model have been improved by using Association Rules to fill missing values.

Key words

MVNO, Mobile Virtual Network Operator, data warehouse, data mining, data mart, support vector machines, association rules, pentaho kettle, pentaho data integration, pentaho data mining, pentaho weka, pentaho bi server, payment behavior, risk

Agradecimientos

A todas las personas que ayudaron y apoyaron. ¡Gracias!

Índice general

Índice de figuras	XI
Índice de cuadros	XIV
1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Objetivos y enfoque	2
1.3. Metodología	2
2. Introducción a las bases de datos	5
2.1. Breve reseña histórica	5
2.2. Modelos de datos	7
2.3. MySQL	8
2.3.1. Arquitectura de MySQL	8
2.3.2. Motores de almacenamiento en MySQL	9
2.3.3. Tipos de datos en MySQL	10
2.3.4. Claves e índices	12
2.3.5. Particiones	14
2.3.6. El comando EXPLAIN	14
3. Introducción a <i>Data Warehousing</i>	17
3.1. Definición y funciones	17
3.2. Arquitectura de un Data Warehouse	18
3.2.1. Sistemas operacionales	18
3.2.2. <i>Data Staging Area</i>	19
3.2.3. <i>Data Presentation Area</i> (DPA)	22
3.3. Metodologías de desarrollo en <i>Data Warehouse</i>	22
3.3.1. Metodologías <i>top-down</i> y <i>bottom-up</i>	22
3.3.2. Toma de requisitos	22

4. Introducción a <i>Data Mining</i>	27
4.1. Definición y aplicaciones	27
4.2. Preparación de datos	28
4.2.1. Inconsistencias	28
4.2.2. <i>Outliers</i>	28
4.2.3. Datos incompletos	30
4.2.4. Excesiva granularidad	31
4.2.5. Atributos de tipos no admitidos	32
4.2.6. <i>Datasets</i> no balanceados	32
4.3. SVM (<i>Support Vector Machine</i>)	33
4.4. Reglas asociativas	37
4.5. Evaluación de rendimiento	38
4.5.1. División del <i>dataset</i>	38
4.5.2. Medidas de rendimiento	39
5. Sistemas empleados en un Operador Móvil Virtual	41
5.1. Definición y peculiaridades de un MVNO (<i>Mobile Virtual Network Operator</i>)	41
5.2. Arquitectura de sistemas	42
5.2.1. Servidores	42
5.2.2. Bases de datos	43
5.2.3. CRM y gestión de incidencias	46
5.3. Procesos	46
5.3.1. Alta de cliente	46
5.3.2. Facturación	48
5.3.3. Cambios de titular	48
6. Estado del arte en predicción de impagos en telefonía	49
7. Diseño del <i>data warehouse</i>	51
7.1. NDS (<i>Normalized Data Store</i>)	51
7.1.1. <i>cliente_</i>	52
7.1.2. <i>prov_</i>	54
7.1.3. <i>factura_</i>	54
7.1.4. <i>cdr_</i>	54
7.1.5. <i>info_</i>	54
7.2. DDS (<i>Dimensional Data Store</i>)	54
7.2.1. Línea	55
7.2.2. Factura	56
7.2.3. CDR	57

8. Diseño del ETL	65
8.1. Conceptos básicos de Pentaho Data Integration	65
8.2. Instalación y configuración de Pentaho Data Integration	65
8.3. Implementación del ETL	68
8.3.1. NDS	69
8.3.2. DDS	84
8.3.3. Reconstrucción del histórico	88
8.3.4. Exportación a Weka	88
9. Diseño del sistema de <i>data mining</i>	91
9.1. Conceptos básicos de Pentaho Data Mining	91
9.1.1. GUI	91
9.1.2. Weka como librería externa	93
9.2. Sistema implementado	95
9.2.1. Lectura del conjunto de datos, aleatorización y división	95
9.2.2. Pre-procesado	95
9.2.3. Optimización de parámetros del clasificador	96
9.2.4. Evaluación	97
10.Resultados	99
10.1. Configuración de los experimentos	99
10.2. Modelo basado en SVM	99
10.3. Modelo basado en SVM con imputación de valores desconocidos mediante reglas asociativas	100
10.4. Modelo basado en bosques aleatorios	101
10.4.1. Discusión de los resultados	103
11.Conclusiones y trabajo futuro	105
Glosario de acrónimos	109
Bibliografía	111
A. Código empleado	117
A.1. <i>User defined java class</i> : Encontrar tipo documento REGEX	117
A.2. <i>Execute SQL script</i> Populate tables	119
A.3. <i>Table input</i> Table input cliente	120
A.4. <i>Execute SQL script</i> _temp_fact_ant y _temp_resumen_pp	121
A.5. <i>Table input</i> en la transformación fact_factura	121

A.6. <i>Table input</i> Input DDS de la transformación <code>weka_cliente</code>	122
A.7. <i>Modified Java Script Value Missing</i> values ->? de la transformación <code>weka_cliente</code>	123
A.8. Programa Java para la evaluación de las SVM	124
A.9. Programa Java para completar datos desconocidos mediante Reglas Asociativas .	131
B. Presupuesto	137
C. Pliego de condiciones	139

Índice de figuras

2.1. Evolución de la tecnología de base de datos	6
2.2. Arquitectura lógica de un servidor MySQL	8
3.1. Esquema básico de un <i>data warehouse</i>	18
3.2. Esquema básico de un <i>Data Staging Area</i>	19
3.3. Esquema básico de un <i>star schema</i>	25
3.4. Esquema básico de un <i>outrigger</i>	25
3.5. Jerarquía de modelos.	25
4.1. Esquema de descubrimiento de conocimiento en bases de datos	27
4.2. Posibles fronteras de decisión	33
4.3. Frontera de decisión y margen de un clasificador SVM ante datos separables. . .	35
4.4. Frontera de decisión y margen de un clasificador SVM ante datos no separables. .	36
5.1. Elementos de un servicio de comunicaciones móviles	41
5.2. Proceso de alta de un cliente	47
7.1. Esquema de las tablas de tipo <code>cliente_</code>	53
7.2. Esquema de las tablas de tipo <code>prov_</code>	53
7.3. Esquema de las tablas de tipo <code>factura_</code>	55
7.4. Esquema de las tablas de tipo <code>cdr_</code>	58
7.5. Esquema de las tablas de tipo <code>info_</code>	59
7.6. El <i>star schema</i> Línea	60
7.7. El <i>star schema</i> Factura	62
7.8. El <i>star schema</i> CDR	63
8.1. Selección de tipo de repositorio.	66
8.2. Selección de base de datos.	66
8.3. Parámetros avanzados del servidor.	67
8.4. Definición del repositorio.	67
8.5. Las tablas del repositorio han sido creadas correctamente.	68
8.6. Creación de nueva conexión a base de datos.	68

8.7. <i>Job</i> cliente_datos	69
8.8. Transformación cliente_datos	70
8.9. <i>Dimension lookup/update</i>	72
8.10. Transformación cliente_linea	73
8.11. Transformación info_producto	74
8.12. <i>Job</i> cliente_pioneros	74
8.13. Transformación cliente_pioneros_tablas	74
8.14. Transformación cliente_pioneros	75
8.15. Transformación cliente_pioneros_monedero	75
8.16. <i>Job</i> cdr_carga_inicial	76
8.17. Transformación cdr_carga_inicial_variable	76
8.18. Transformación carga_cdr_inicial	77
8.19. <i>Job</i> cdr_carga_incremental	77
8.20. Transformación cdr_carga_incremental_filas	78
8.21. Transformación cdr_carga_incremental	79
8.22. Transformación cdr_carga_incremental_carga_id	80
8.23. <i>Job</i> factura	80
8.24. Transformación factura_cliente	81
8.25. Transformación factura_cliente_pioneros	82
8.26. Transformación factura_linea	83
8.27. Transformación factura_linea_consumos	83
8.28. <i>Job</i> dim_estaticas	84
8.29. <i>Job</i> dim_fecha	85
8.30. Transformación dim_fecha	85
8.31. <i>Job</i> fact_cliente	86
8.32. Transformación fact_cliente	86
8.33. <i>Job</i> fact_factura	87
8.34. Transformación fact_factura	88
8.35. Transformación weka_cliente	89
9.1. Pantalla inicial de WEKA	92
9.2. Módulo de preprocesado del <i>Explorer</i>	92
9.3. Módulo de clasificación del <i>Explorer</i>	93
9.4. <i>Run configurations</i> en Eclipse IDE	94
9.5. <i>Package explorer</i> en Eclipse IDE	94
9.6. Esquema del sistema a través del cuál se ha evaluado el modelo	95

10.1. Curva ROC del modelo de bosques aleatorios. El eje horizontal es la tasa de falsos positivos, y el eje vertical es la tasa de verdaderos positivos.	103
11.1. Pantallazo de Saiku Analytics, un <i>plug-in</i> de Pentaho BI Server conectado al DDS implementado.	106

Índice de tablas

2.1. Ejemplo de tabla en modelo relacional	7
2.2. Tipos enteros en MySQL	11
2.3. Recomendaciones de uso de cada tipo de cadena de texto	12
2.4. EXPLAIN antes de optimizar consulta.	16
2.5. EXPLAIN tras optimizar consulta.	16
3.1. Ejemplo de paquete de información.	23
4.1. Ejemplo de transacciones binarizadas.	37
4.2. Ejemplo de matriz de confusión.	39
6.1. Resumen de estudios sobre insolvencia en telecomunicaciones móviles	50
7.1. Matriz de dimensiones conformadas	56
7.2. <i>Information package</i> de Línea (parte 1)	57
7.3. <i>Information package</i> de línea (parte 2)	58
7.4. <i>Information package</i> de Factura (parte 1)	59
7.5. <i>Information package</i> de factura (parte 2)	61
7.6. <i>Information package</i> de CDR	61
10.1. Resultados de MultiSearch (<i>3-fold cross validation</i>)	100
10.2. Evaluación del modelo con los parámetros encontrados con MultiSearch frente al conjunto de test.	100
10.3. Resultados de MultiSearch (<i>3-fold cross validation</i>)	101
10.4. Evaluación del modelo con los parámetros encontrados con MultiSearch frente al conjunto de test.	101
10.5. Evaluación del modelo de bosques aleatorios con rellenado de valores desconocidos por MC.	102
10.6. Evaluación del modelo de bosques aleatorios con rellenado de valores desconocidos mediante reglas asociativas.	102

1

Introducción

1.1. Motivación del proyecto

El fraude en telecomunicaciones es el abuso deliberado de las redes de voz y datos con la intención de reducir o evadir el pago por los servicios utilizados. Según el informe *Global Fraud Loss Survey* del año 2013, que tiene en cuenta los datos de fraude de un alto número de operadores a nivel mundial, las pérdidas por este motivo ascendieron a 46.300 millones de dólares, suponiendo un 2.09 % de los ingresos. [1]

Si bien este documento ya refleja una tendencia ascendente en este tipo de conductas, es habitual que las nuevas compañías sean más vulnerables al fraude que las ya establecidas, ya que la mayor parte de las líneas fraudulentas son nuevas altas. El gran volumen de datos disponible en la actualidad ha hecho que los métodos clásicos de detección de fraude, basados en el análisis aislado de coste y duración de CDRs (*Call Detail Records*), hayan quedado obsoletos. Por tanto, es conveniente recurrir a métodos más sofisticados de detección automática basados en técnicas de *Data Mining*, entendiendo como tal el proceso de exploración y análisis automático o semi-automático de grandes cantidades de datos para descubrir patrones y reglas significativas, empleando técnicas estadísticas, matemáticas y de reconocimiento de patrones.

Existe una gran cantidad de métodos de predicción y clasificación, cuyo desempeño depende de factores como el tamaño de la base de datos, los tipos de patrones existentes en los datos, si éstos cumplen o no ciertas asunciones que el modelo tiene en cuenta, el nivel de ruido, y el fin particular del análisis. Pueden ser aplicados tanto a grandes bases de datos, como a las generadas por un operador móvil.

Hay que tener en cuenta que las bases de datos OLTP (*OnLine Transaction Processing*), usadas para almacenar transacciones individuales, no son adecuadas para análisis globales complejos. Es ahí donde aparece la necesidad de la creación de un *Data Warehouse*, un repositorio centralizado de información precisa, fiable, creíble y comprensible, a partir del cual es posible almacenar información con datos históricos para los diversos departamentos, y visible a través de *Data Marts*; buscar patrones entre los clientes mediante técnicas de *Data Mining* y realizar predicciones que puedan ser posteriormente cargadas de vuelta al sistema.

La información que contiene un *Data Warehouse* puede provenir de bases de datos de CRM (*Customer Relationship Management*), logística, proveedores, facturación, etc.; hojas de cálculo e incluso datos obtenidos de redes sociales. Todas estas fuentes de datos han de ser integradas

mediante procedimientos ETL (*Extract-Transform-Load*) o ELT (*Extract-Load-Transform*), con el fin de normalizar la información para su uso por parte de los usuarios, ya sea directamente o en forma de KPIs (*Key Performance Indicators*).

Una vez se cuenta con la información correctamente procesada se pueden aplicar algoritmos de *data mining*. En concreto, este problema se puede plantear como un problema de clasificación que puede ser resuelto mediante aprendizaje supervisado, ya que se cuenta con datos de líneas fraudulentas anteriores, consiguiendo etiquetar a cada línea telefónica como fraudulenta o no-fraudulenta. Sin embargo, aunque este tipo de método puede ser bueno a la hora de detectar conductas fraudulentas similares a las ya sufridas, es posible que no se adapte a nuevos patrones de fraude. Por tanto, se experimentará combinando los algoritmos de clasificación con un método de aprendizaje no supervisado como las reglas asociativas, evaluando las posibles mejoras del sistema.

Todo el proyecto formará parte de la solución BI (*Business Intelligence*) de la compañía. El objetivo final de este tipo de sistemas es ser empleados como DSS (Decision Support System), que pueden ser usados por los gerentes y ejecutivos para tomar decisiones fundamentadas en datos accesibles, objetivos y fiables, sin la necesidad de solicitar al Departamento de Sistemas informes ad-hoc.

1.2. Objetivos y enfoque

El proyecto tiene como objetivo principal proporcionar un método de predicción de impagos en un operador móvil. Para ello, se diseñará e implementará un *Data Warehouse*, teniendo en mente la necesidad de que sea lo suficientemente flexible, ya que será utilizado para otras aplicaciones dentro de la solución BI de la empresa. Este repositorio de datos contendrá información proveniente de las bases de datos de la compañía (CRM, logística, facturación. . .) y, de no existir la información en la base de datos operacional, de hojas de cálculo Excel que provean los distintos departamentos (finanzas, operaciones). Se trabajará con un *Data Mart* específico para detección de impagos.

Una vez se cuente con una fuente fiable de datos, se procesará la información mediante los algoritmos de clasificación que mejor resultado han dado teniendo en cuenta la literatura disponible. Por último, se tratará de mejorar dichos algoritmos con reglas asociativas, y se compararán los resultados.

1.3. Metodología

El proyecto se va a realizar en el Departamento de Sistemas de la operadora móvil virtual República Móvil (República de Comunicaciones Móvil, S.L.). Por tanto, se contará con datos reales que serán debidamente procesados para mantener la confidencialidad de los clientes. Dado que el proyecto forma parte de la solución BI de la compañía, se utilizarán herramientas incluidas en la plataforma BI de código abierto Pentaho: Kettle, Weka y Kettle.

La información se tomará de bases de datos MySQL, por lo que se emplearán consultas y procedimientos almacenados para acceder y procesar los datos.

Se empleará Pentaho Data Integration (Kettle) para obtener y procesar la información y replicarla en el *Data Warehouse*. Será entonces cuando se utilice Pentaho Data Mining (Weka) para realizar la detección de fraude, añadiendo los módulos necesarios para combinar clasificación con reglas asociativas, utilizando lenguaje Java. El plan de trabajo está basado en la metodología CRISP-DM y se resume en:

1. Estudio y aprendizaje previo:

- Estado del arte: se analizarán las tendencias actuales en cuanto a los métodos de almacenamiento de información y análisis en data warehouse y data mining aplicados a la gestión de fraude.
- Sistemas: se analizarán a fondo las bases de datos y sistemas operacionales de República Móvil.
- Software: se aprenderá a realizar consultas y procedimientos almacenados en MySQL, así como programas en Java o PHP que controlen el flujo de éstos, y a utilizar la suite BI Pentaho para realizar integración de datos, informes y data mining.

2. Obtención del conjunto de datos a analizar:

- Diseño e implementación del *Data Warehouse*, teniendo en cuenta que la detección de fraude es solo una de sus aplicaciones.
- Integración y pre-procesado de datos: unir todas las fuentes de datos y procesarlas de forma que se obtenga una única visión global fiable y depurada.
- Diseño e implementación de Data Marts según las especificaciones de departamentos, así como uno dedicado a fraude que se analizará mediante *Data Mining*.
- Muestreo: reducir el volumen de datos en caso de ser necesario, y dividir los conjuntos de entrenamiento, validación y test.

3. Aplicación de data mining al conjunto de datos:

- Elección del algoritmo de clasificación a emplear según el estado del arte.
- Combinación de éste con reglas asociativas.
- Evaluación del desempeño de éstos con distintos parámetros.
- Interpretación de los resultados.

4. Elección y despliegue:

- Elección del algoritmo que mejores resultados ha dado.
- Despliegue del modelo: se usará sobre datos reales, volcando los resultados al *Data Warehouse*, para su uso como DSS.

2

Introducción a las bases de datos

Hoy en día, las bases de datos son componentes esenciales de cualquier proyecto. Por ejemplo, los portales web más importantes, como Google o Amazon, u otros más pequeños, proveen al visitante de información que almacena en una base de datos, mientras que las corporaciones custodian en ellas toda la información importante.

En realidad, las bases de datos no son más que una colección de información que lleva existiendo un cierto período de tiempo, normalmente varios años. Al hablar de una base de datos nos estamos refiriendo, en general, a un conjunto de datos controlado por un DBMS (*Database Management System*, Sistema de gestión de base de datos), el encargado de hacer posible la creación de nuevas bases de datos que almacenen grandes cantidades de información durante largos períodos de tiempo, permitiendo a múltiples usuarios realizar consultas para extraerla o modificarla de manera simultánea. El DBMS ejerce, por tanto, el papel de interfaz entre la base de datos y las aplicaciones y usuarios que la utilizan. [2]

2.1. Breve reseña histórica

Las bibliotecas y registros existen desde la Antigüedad. Entonces se utilizaban para almacenar información sobre censos y cosechas, aunque los procesos de búsqueda manual resultaban lentos y poco eficaces, además de ser muy vulnerables a errores. [3] Tras trabajar en el laborioso censo manual de Estados Unidos de 1880, Herman Hollerith diseñó una tabuladora que fue empleada en el censo de 1890, consiguiendo reducir de diez años a entre seis semanas y tres años (según la fuente) el tiempo necesario para realizarlo. Este fue el primer sistema de procesamiento de información que pudo reemplazar por completo al papel y al lápiz, y tuvo gran aceptación internacional, hasta el punto de que la compañía creada por Hollerith terminó convirtiéndose en IBM tras una fusión. [4]

Más adelante, con el desarrollo de los primeros computadores, el concepto de base de datos pasó a estar ligado con la informática. El apogeo de las cintas magnéticas en la década de 1950 creó la posibilidad de automatizar el procesamiento de información de forma secuencial. [3] Las primeras aplicaciones informáticas permitían realizar cálculos y almacenar datos en sistemas basados en ficheros, hasta que quedaron obsoletos debido a las nuevas necesidades.

En la década de los 60, el gobierno de J. F. Kennedy inició el proyecto Apolo, con el aterrizaje

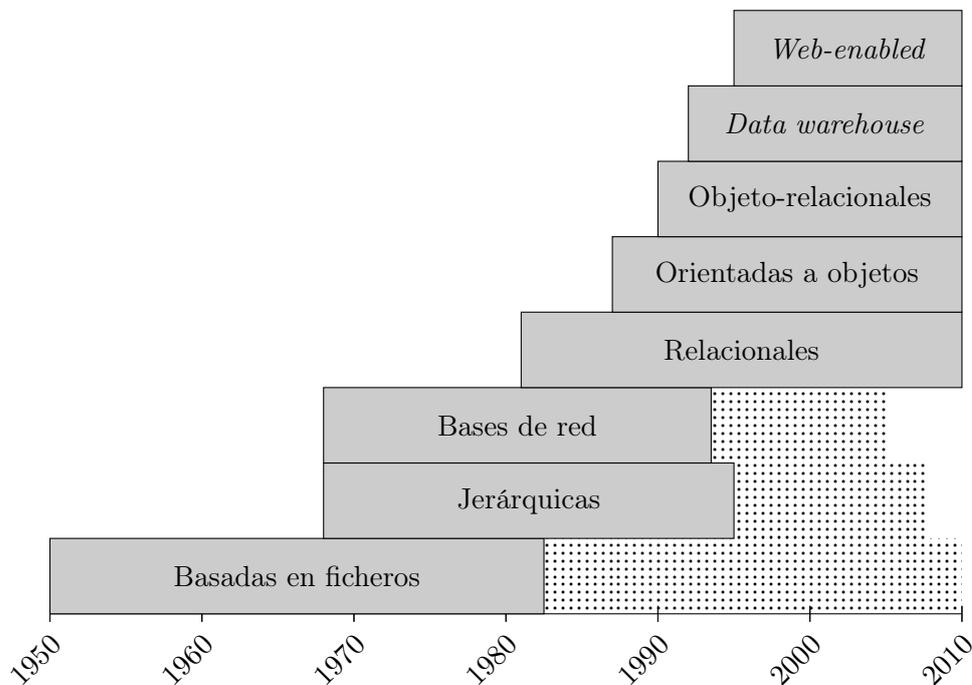


Figura 2.1: Evolución de la tecnología de base de datos [5]

humano en la Luna como objetivo. Se esperaba que el proyecto generara grandes volúmenes de datos que resultaban imposibles de manejar para un sistema basado en ficheros, por lo que la North American Aviation, principal contratista del proyecto, desarrolló un sistema jerárquico denominado *Generalized Update Access Method* (GUAM). Este sistema se convirtió más adelante en *Information Management System* (IMS), en cuyo desarrollo participó IBM, que por otra parte estaba trabajando con American Airlines para que otro sistema, SABRE, que permitía la reserva simultánea en las aerolíneas, viera la luz.

Mientras tanto, en General Electric, Charles Bachman había estado diseñando *Integrated Data Store* (IDS), acuñando el concepto de base de datos de red. Este sistema, una generalización de GUAM, se convirtió en un estándar durante los años 60, e hizo que Bachman fuera galardonado con el Premio Turing el año 1973.

En 1970, Edgar Codd escribió un artículo en el que proponía un modelo de bases de datos relacionales. La influencia de éste sigue patente el día de hoy: entre las ideas clave estaban el uso de tablas como única representación de datos, y el empleo del lenguaje SQL (*Structure Query Language*) para acceder a éstos a alto nivel. Como resultado de este trabajo, a Codd se le concedió el Premio Turing de 1981.

A pesar de su interés, el modelo relacional no se empezó a utilizar en la práctica debido a que su rendimiento era peor que el de los otros modelos existentes. Fue en los años 80, con el desarrollo de *Structured Query Language / Database System* (SQL/DS) por parte de IBM cuando esta tecnología consolidó su posición en el mercado, dando lugar a múltiples sistemas que, en mayor o menor medida, se siguen usando en la actualidad, como ORACLE, IBM DB2, Microsoft Access o Corel Paradox.

Los años 90 siguieron llenos de actividad: al principio de la década, el lenguaje SQL fue estandarizado por ANSI (American National Standards Institute), mientras que la posibilidad de emplear las bases de datos de forma concurrente se popularizó mediante el uso de las transacciones. También comenzaron a surgir nuevos modelos con el objetivo de tratar datos cada vez más complejos, como las bases de datos orientadas a objetos y las bases de datos objeto-relacionales,

así como los primeros *Data Warehouse* que surgieron al final de la década. [5]

2.2. Modelos de datos

Uno de los conceptos fundamentales en bases de datos es el de *modelo de datos*, que describe la estructura de los datos, las operaciones definidas sobre éstos y las restricciones que deben cumplir. Los dos tipos de modelo prominentes el día de hoy son el modelo relacional y el modelo de datos semi-estructurado. [2]

Modelo relacional

El modelo relacional está basado en tablas. La estructura consta de columnas, que definen una serie de atributos (por ejemplo, MSISDN, fecha de activación y estado), y filas, cada una de las cuáles se corresponde con una entrada. Las operaciones definidas vienen dadas por el álgebra relacional que, por ejemplo, permite seleccionar y unir tablas mediante sentencias SQL. Por último, entre las restricciones, es posible definir claves únicas o enumeraciones. De esta forma, se podría hacer que el estado de la línea únicamente pueda tomar los valores 'Activo' o 'Baja', o que un mismo MSISDN aparezca una única vez en la tabla.

MSISDN	Fecha de activación	Estado
512453865	20/01/2013	Activo
526875643	30/05/2014	Activo
548623158	20/01/2013	Baja

Tabla 2.1: Ejemplo de tabla en modelo relacional

Entre los principales DBMS relacionales actuales se encuentran Microsoft SQL Server, Oracle Database y MySQL, que es el sistema empleado en este proyecto. Si bien todos trabajan con lenguaje SQL, suele haber diferencias entre ellos. Por ejemplo, Oracle cuenta con la operación `OUTER JOIN`, mientras que en MySQL debe ser simulada mediante un `LEFT JOIN`, un `RIGHT JOIN` y una cláusula `WHERE`.

Modelo semi-estructurado

El modelo semi-estructurado está basado en árboles, y se emplea en formatos como XML y JSON. En general, las operaciones definidas se limitan a seguir caminos de un elemento a una serie de elementos anidados en éste de forma recursiva. Entre las restricciones, es posible definir el tipo de dato asociado a una determinada etiqueta.

Código 2.1: Ejemplo de modelo semi-estructurado XML

```
1 <Lista_Lineas>
2   <Linea>
3     <MSISDN>512453865</MSISDN>
4     <Fecha_activacion>20/01/2013</Fecha_activacion>
5     <Estado>Activo</Estado>
6   </Linea>
7   <Linea>
8     <MSISDN>526875643</MSISDN>
9     <Fecha_activacion>20/01/2013</Fecha_activacion>
```

```

10         <Estado>Activo</Estado>
11     </Linea>
12     <Linea>
13         <MSISDN>512453865</MSISDN>
14         <Fecha_activacion>30/05/2014</Fecha_activacion>
15         <Estado>Activo</Estado>
16     </Linea>
17     <Linea>
18         <MSISDN>548623158</MSISDN>
19         <Fecha_activacion>20/01/2013</Fecha_activacion>
20         <Estado>Baja</Estado>
21     </Linea>
22 </Lista_Lineas>

```

Entre sus múltiples aplicaciones, XML es el formato utilizado por Pentaho a la hora almacenar transformaciones y fuentes de datos.

2.3. MySQL

MySQL es un DBMS relacional multihilo de código abierto creado en 1995, y liberado al público general bajo licencia GNU-GPL el año 2000. MySQL AB, su propietaria, es ahora subsidiaria de Sun Microsystems (que desarrolla Oracle Database), y estima que hay más de 6 millones de instalaciones en todo el mundo. Además de poder usarse gratuitamente, es robusta, goza de buen rendimiento y está llena de posibilidades. [6]

Entrar a fondo en las características de este sistema podría llevar cientos de páginas, por lo que esta sección únicamente nos centraremos en algunos de los aspectos más importantes a la hora de conseguir un *Data Warehouse* fiable y de óptimo rendimiento.

2.3.1. Arquitectura de MySQL

MySQL sigue el modelo cliente-servidor, por lo que los clientes tienen que pasar por los procesos correspondientes, tales como gestión de conexiones, autenticación, seguridad, etc. El encargado de esto es el bloque de *Gestión de conexiones / hilos*.

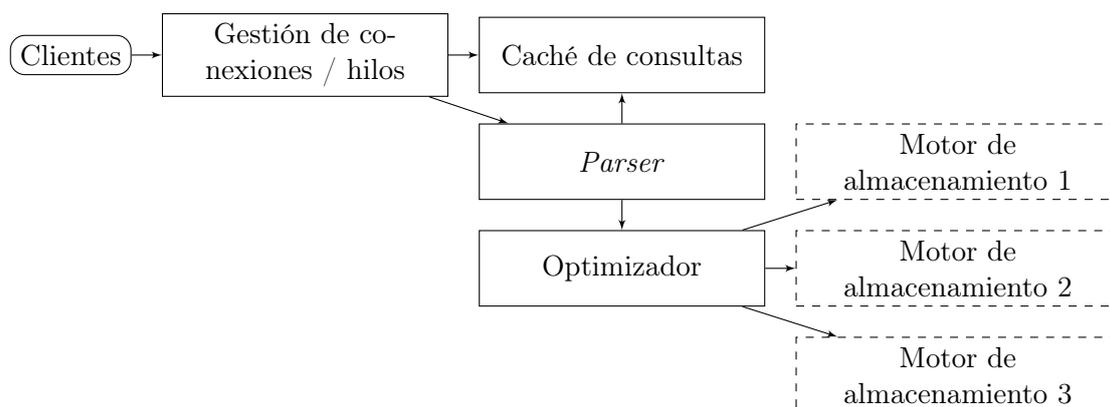


Figura 2.2: Arquitectura lógica de un servidor MySQL [7]

En la segunda capa tienen lugar los procesos de interpretación, análisis, optimización, cachado de consultas, así como las funciones integradas (matemáticas, fechas, etc.). También

tiene aquí lugar toda la funcionalidad inherente a todos los motores de almacenamiento, como los procedimientos almacenados, *triggers* o vistas.

En la tercera y última capa aparecen los motores de almacenamiento, los responsables de guardar y recuperar todos los datos almacenados en MySQL. El sistema se encarga de proporcionar una interfaz que esconde las diferencias entre los distintos motores, de forma que las consultas pueden ser realizadas de forma transparente. Los motores no interpretan SQL ni se comunican entre ellos, únicamente se encargan de responder peticiones del servidor.

Control de concurrencia

Un sistema de base de datos debe ser capaz de gestionar órdenes de lectura y escritura simultáneas. MySQL se encarga de esto tanto a nivel de servidor como a nivel de motor de búsqueda, por lo que el comportamiento dependerá de la elección de éste.

Para evitar problemas de inconsistencia de datos se definen dos tipos de bloqueo:

- **Bloqueo de lectura**, que permite el acceso concurrente a los mismos datos de forma no exclusiva.
- **Bloqueo de escritura**, que permite un único acceso concurrente a los mismos datos, de forma mutuamente exclusiva.

Además del tipo de bloqueo debe definirse la granularidad de éste, siendo posible bloquear una única fila o la tabla completa. Existe, por tanto, un compromiso entre la seguridad de los datos y los recursos necesarios para gestionar los bloqueos. Cuanto mayor sea la concurrencia, mayor será el coste computacional.

La mayoría de los motores transaccionales de MySQL soportan MVCC (*Multiversion Concurrency Control*), que evita la necesidad de bloqueo en una buena parte de los casos, consiguiendo un aprovechamiento más eficiente de los recursos.

Transacciones

Una transacción es un grupo de secuencias SQL tratadas de forma atómica, como una unidad de trabajo. Su objetivo es que los cambios se apliquen únicamente si todas las operaciones han resultado exitosas (con un comando **COMMIT**) mientras que, en caso de error, sea posible recuperar los datos anteriores como si nada hubiera pasado (**ROLLBACK**).

Los motores de almacenamiento pueden soportar o no transacciones y, de hacerlo, deben superar el test ACID – *Atomic, Consistent, Isolated, Durable*. Esto es muy costoso a nivel de CPU, por lo que la elección de motor de almacenamiento puede depender fuertemente de si es necesario soportar transacciones.

2.3.2. Motores de almacenamiento en MySQL

Una buena parte de la flexibilidad de MySQL reside en la posibilidad de escoger el motor de almacenamiento empleado en cada tabla. Los motores de propósito general incluidos en la distribución estándar de MySQL son:

- **InnoDB**, un motor transaccional de alto rendimiento y fiable contra errores. Bloquea a nivel de fila, y soporta MVCC y *Foreign Keys* (claves foráneas) para mantener integridad referencial, así como índices agrupados¹. [8]

¹Se introducirán estos conceptos el capítulo de *data warehouse*.

- **MyISAM**, el motor no transaccional que fue el empleado por defecto hasta la versión 5.1. El control de concurrencia es menos avanzado que el de InnoDB: bloquea a nivel de tabla, por lo que su rendimiento es menor en situaciones que conlleven mucha carga de lectura y escritura simultáneas. Sin embargo, permite ejecutar sentencias `INSERT` concurrentes. Al no cumplir ACID, no es tan seguro ante errores, pero es más simple y necesita menos recursos.

La discusión en cuanto al uso de uno u otro está bastante abierta. Según [7], InnoDB es el motor más útil en la mayoría de los casos, y se recomienda escogerlo por defecto salvo que haya una buena razón para no hacerlo (incluso cuando no sean necesarias las transacciones), afirmando que el bloqueo a nivel de tabla de MyISAM tiene serias implicaciones en el rendimiento, aunque puede ser útil para tablas no muy grandes de sólo lectura. Sin embargo, ni siquiera en esos casos está garantizado que MyISAM sea más rápido, ya que InnoDB puede utilizar índices agrupados.

Otros usuarios [9] afirman que los recursos extra que requiere InnoDB para cumplir ACID son razón suficiente para usar MyISAM siempre que no sean necesarias las transacciones. Sin embargo, esto complica la recuperación en caso de errores del sistema, ya que InnoDB se podría reconstruir a partir de los *logs*, mientras que una tabla MyISAM tendría que ser recuperada por completo, recreando tablas e índices.[10]

Además de éstos, existen otros motores con aplicaciones más particulares, entre los que se encuentran:

- **Archive**, un motor optimizado para inserciones rápidas que almacena filas comprimidas. Únicamente permite sentencias `SELECT` e `INSERT`, utilizando bloqueo a nivel de fila. No soporta índices, por lo que cualquier consulta `SELECT` debe recorrer toda la tabla. Esto hace que sean útiles para mantener logs o adquirir datos.
- **Memory**, un motor que almacena los datos en memoria volátil. Es útil cuando se necesita acceso rápido a datos que no cambian o que no importa perder tras un reinicio, como tablas temporales que acumulan resultados. Son hasta un orden de magnitud más rápidas que MyISAM, pero también bloquean a nivel de tabla. MySQL intenta² utilizar este tipo de tabla cada vez que una consulta requiera el uso de una tabla intermedia.

Un factor a tener en cuenta en la elección de motor de almacenamiento es la complejidad añadida en las interacciones internas cuando varios motores distintos entran en juego.

2.3.3. Tipos de datos en MySQL

La elección del tipo de dato de cada columna puede ser muy relevante en cuanto a rendimiento y espacio ocupado por la base de datos. En primer lugar, debe escogerse el tipo general (decidir si el dato es un número, una cadena de texto o una fecha) y, desde ahí, concretar el tipo de dato de MySQL.

Schwartz et al. [7] proponen tres recomendaciones generales acerca de la elección:

- *Cuanto más pequeño, mejor.* Así, se ocupará menos espacio en disco y se requerirán menos recursos en general. Sin embargo, hay que tener cuidado con definir un rango suficiente para evitar desbordamientos.
- *Cuanto más simple, mejor.* Por ejemplo, es más fácil comparar dos datos `INT` que dos datos `CHAR` ya que no va a haber distintos juegos de caracteres que puedan causar pérdidas de rendimiento.

²No podrá hacerlo con tablas demasiado grandes o que contengan datos tipo `BLOB` o `TEXT`.

- *NULL debe evitarse en la medida de lo posible.* Las columnas que pueden contener valores nulos requieren un tratamiento especial cuando se indexan, por lo que habrá una pequeña ganancia en rendimiento al declarar las columnas que no requieran nulos con la propiedad `NOT NULL`. Por otra parte, al utilizar `NULL` en MyISAM se ocupa más espacio en disco, incluso haciendo que una fila de tamaño fijo pase a tenerlo variable. InnoDB, por el contrario, almacena el valor nulo con un único bit.

A continuación se explorarán los tipos generales de datos, y se estudiarán las opciones que ofrece MySQL para su tratamiento.

Números enteros

La elección del tipo de dato para números enteros debe hacerse según el rango necesario. Cada uno de ellos ocupa N bits, y tendrá un rango de $-2^{(N-1)}$ a $2^{N-1} - 1$, o de 0 a $2^N - 1$ si la columna tiene la propiedad `UNSIGNED`.

Tipo de dato	N (bits)
TINYINT	8
SMALLINT	16
MEDIUMINT	24
INT	32
BIGINT	64

Tabla 2.2: Tipos enteros en MySQL

Merece la pena señalar que la sintaxis tipo `INT(11)` a la hora de la definición no tiene efecto sobre el tamaño o el rango de la columna. Únicamente afecta a su representación.

Números reales

En general, se recomienda utilizar los tipos de coma flotante (`FLOAT` y `DOUBLE`) ya que las operaciones se realizan de forma nativa en el Sistema Operativo, ganando en velocidad frente al tipo de coma fija (`DECIMAL`), cuyas operaciones deben realizarse a nivel de DBMS.

Sin embargo, `DECIMAL` es útil a la hora de almacenar y representar números fraccionales exactos, como es el caso de cantidades monetarias, ya que MySQL no tiene definido un tipo `MONEY` como otros DBMS. La Junta de Andalucía [11] recomienda su uso³, mientras que Frank Rietta sugiere seguir los *Generally Accepted Accounting Principles* de la agencia gubernamental estadounidense Federal Accounting Standards Advisory Board, empleando `DECIMAL(13,4)`⁴. Por otra parte, [7] sugiere multiplicar el valor por 100 y utilizar `BIGINT`, ganando en velocidad de cálculo.

Texto

MySQL puede almacenar cadenas de texto de longitud fija (`CHAR`) o variable (`VARCHAR`). Las segundas necesitan hasta 2 Bytes extra (según la longitud) para almacenar la longitud, y

³Recomiendan el uso de la clase `java.math.BigDecimal` en Java, que según el manual debe mapearse a `DECIMAL` en MySQL. [12]

⁴El primer argumento indica el número total de dígitos permitidos, mientras que el segundo indica el número de dígitos permitidos tras la coma decimal. Al contrario que en `INT`, en este caso sí afecta a la hora del almacenamiento.

pueden dar problemas de rendimiento con sentencias `UPDATE`, ya que la fila cambiaría de tamaño, causando comportamientos variables según el motor de almacenamiento. Las recomendaciones de uso de cada tipo se recogen en la tabla 2.3.

VARCHAR	CHAR
La tabla se actualiza muy raramente.	La tabla se actualiza frecuentemente.
La longitud máxima de la columna es mucho mayor que la longitud media.	La longitud del campo es aproximadamente fija.
Se utiliza un juego de caracteres complejo (como UTF-8), cuyos caracteres ocupan un espacio variable.	La longitud es muy corta.

Tabla 2.3: Recomendaciones de uso de cada tipo de cadena de texto [7]

Por otra parte, existen los tipos `TEXT` y `BLOB`, que permiten almacenar texto como caracteres o como una secuencia binaria, respectivamente. Estos tipos no se pueden indexar al completo, y no son soportados por el motor de almacenamiento `MEMORY`.

En general, [7] recomienda evitar el empleo de cadenas de texto como identificadores, ya que suelen ocupar más y ser más lentos que números enteros.

Enumeraciones

El tipo `ENUM` permite definir una serie de constantes, cuyo valor puede tomar la columna. Es útil porque almacena enteros en vez de texto, pero es necesaria la compleja operación `ALTER TABLE` para cambiar el valor de las constantes. Además, aunque un `JOIN` entre dos campos `ENUM` es muy rápido, es muy costoso efectuar un `JOIN` entre `ENUM` y `VARCHAR`, siendo más rápida esta operación entre dos campos `VARCHAR`.

Fecha y hora

Existen dos tipos para almacenar fecha y hora. El tipo `DATETIME` tiene un rango desde el año 1001 hasta el 9999, tomando 8 bytes. Por otro lado, `TIMESTAMP` está definido desde el año 1970 hasta 2038, y ocupa 4 bytes. Ya hay abierta una tarea de desarrollo para la versión 7.0 donde se pretende aumentar el rango de este tipo de datos.[13]

2.3.4. Claves e índices

La optimización de los índices es probablemente el proceso que mayor mejora produce sobre el rendimiento de las consultas, pudiendo mejorarlo varios órdenes de magnitud. Se encargan de guardar la información ordenada de forma que su recuperación sea más rápida, ya que reducen la cantidad de datos que el servidor debe examinar, ayudan a evitar volver ordenar los datos o usar tablas temporales y convierten operaciones E/S aleatorias en secuenciales.

Los índices más usados, denominados `B-TREE`, están basados en árboles de búsqueda binarios. Facilitan la búsqueda cuando se trata de encontrar un valor concreto, un rango de valores o un prefijo. También es posible utilizar *covering indexes*, índices que contienen todas las columnas de la consulta para la que están preparados. De esta forma, es posible devolver el resultado únicamente leyendo el índice, sin necesidad de acceder al contenido de las filas.

Sin embargo, el mantenimiento de los índices complica la inserción. La estrategia normalmente utilizada en *Data Warehouse*, especialmente cuando se realizan cargas masivas a tablas MyISAM, es la de desactivar los índices antes de cargar datos, realizar la inserción, y reactivar los índices, de forma que el servidor los construya una única vez. Esto ayuda a mantener el árbol de índices compacto y desfragmentado.

A la hora de escoger las columnas que formarán parte de un índice, hay que tener varios factores en cuenta. En primer lugar, los índices deben estar diseñados de acuerdo con las consultas que vayan a ser realizadas sobre la tabla. En concreto, el rendimiento de una cláusula `join` puede aumentar varios órdenes de magnitud si las columnas especificadas están indexadas correctamente.

En segundo lugar, debe prestarse atención al número de filas que componen la tabla a indexar. Además de ocupar espacio en disco, el acceso a un índice consume ciertos recursos. En tablas de tamaño reducido, el coste de acceder al índice puede ser mayor que el de realizar un `FULL SCAN` sobre la tabla.

En tercer lugar, cuando se realizan consultas que buscan valores en varias columnas, es posible utilizar índices compuestos. Al trabajar con índices multi-columna, hay que tener en cuenta que el orden de las columnas afecta al rendimiento del índice, ya que el índice ordena las columnas de izquierda a derecha. Por ejemplo, en una tabla que contenga datos asociados a identificadores de cliente, y los MSISDN (*Mobile Subscriber ISDN (Integrated Services for Digital Network) Number*, el número de teléfono asociado a la tarjeta SIM) de las líneas que éstos hayan contratado, podría seguirse una estrategia de indexación basada en la selectividad de las columnas. En general⁵, un cliente tendrá 1 o más líneas contratadas. Por tanto, el MSISDN será más selectivo, y debería tomar la primera posición del índice.

Sin embargo, MySQL solo utilizará el índice si se realiza un búsqueda que contenga las columnas en orden. Es decir, si creáramos el siguiente índice:

Código 2.2: Ejemplo de índice multicolumna

```
1 CREATE INDEX idx_msisdn_idcli ON tabla (MSISDN, Id_Cliente);
```

Si ejecutáramos las consultas que aparecen en Código 2.3, MySQL utilizaría `idx_msisdn_idcli` para las consultas 1 y 3, pero no para la segunda, que solo busca valores en la segunda columna del índice, sin utilizar la primera. Para solucionar este comportamiento, habría que reescribir la consulta o añadir un índice adicional a la tabla.

Código 2.3: Ejemplos de consulta

```
1 -- Consulta 1
2 SELECT * FROM tabla
3 where MSISDN = '546372876'
4     and Id_Cliente='M987623';
5
6 -- Consulta 2
7 SELECT * FROM tabla
8 where Id_Cliente='M987623';
9
10 -- Consulta 3
```

⁵Un MSISDN podría cambiar de titular y aparecer varias veces, aunque ocurre en una minoría de los casos.

```
11 SELECT * FROM tabla
12 where MSISDN = '546372876';
```

El encargado de escoger los índices a utilizar es el Optimizador, si bien podemos sugerir o forzar el índice empleado mediante *index hints*.

2.3.5. Particiones

Mediante el uso de particiones, MySQL permite definir una tabla lógica como la composición de un cierto número de tablas físicas subyacentes. Gracias a este mecanismo, especialmente útil en tablas de gran tamaño, se puede reducir el número de líneas que el RDBMS debe analizar en una situación de búsqueda. La división se realiza especificando un campo y los rangos de valores a separar en la cláusula **PARTITION BY**, ya sea en el momento de la creación de la tabla, o posteriormente mediante una sentencia **ALTER TABLE**. Si por ejemplo, se particionara una tabla por fecha de registro, al realizar una consulta sobre un rango de fechas, MySQL solo analizaría las particiones definidas para ese rango de fechas. Al contrario que en Oracle, los índices quedan definidos a nivel de partición.

Sin embargo, las particiones tienen algunas restricciones en MySQL. En primer lugar, hay un límite de 1024 particiones por tabla. De acuerdo con [7], esta limitación no resulta demasiado importante ya que, por el coste asociado a analizar qué particion(es) contienen registros relevantes, se recomienda que el número de particiones a mantener sea menor que 200. Además, todas las particiones deben usar el mismo motor de almacenamiento.

En segundo lugar, la clave primaria debe contener todas las columnas utilizadas para definir las particiones.

Por último, las claves foráneas no están soportadas en particiones.

2.3.6. El comando EXPLAIN

A la hora de diseñar una consulta SQL, resulta útil disponer de un mecanismo mediante el cual poder conocer el plan de ejecución decidido por el Optimizador. El comando **EXPLAIN** devuelve una fila por cada tabla consultada, siendo las siguientes columnas las más relevantes:

- **Id**: identifica la posición del **SELECT** dentro de la consulta. Sólo será distinto de 1 si hay *subqueries*.
- **select_type**: especifica si la consulta es **SIMPLE** o compleja y, de serlo, contiene su tipo: **SUBQUERY**, **DERIVED**, **UNION** o **UNION RESULT**.
- **table**: el nombre o alias de la tabla a la que se refiere la fila.
- **type**: describe cómo MySQL va a tratar de encontrar filas en la tabla. Entre los valores posibles destacan, ordenadas de menor a mayor rendimiento:
 - **ALL**: se analizarán todas las filas de la tabla.
 - **index**: se analizará el índice, y se leerá la tabla a partir de él. Si la columna **extra** especifica **using index**, se estará empleando un *covering index*. Esto resultará más barato que leer toda la tabla en el orden especificado por el índice, lo cuál supondría realizar lecturas aleatorias desde el punto de vista del disco.

- **range**: se está barriendo la parte del índice que se corresponde con un rango de valores buscado.
 - **ref**: el índice se compara con un valor de referencia, ya sea constante o proveniente de otra tabla consultada. Solo se puede dar con índices no únicos.
 - **eq_ref**: se realiza una búsqueda en el índice, con la certeza de que se va a encontrar un único valor. Es decir, el índice usado es una clave primaria (PK) o un índice único. MySQL puede optimizar muy bien en este caso, ya que sabe que no tiene que buscar rangos de valores.
- **possible_keys**: lista los índices que podrían emplearse en la consulta.
 - **key**: muestra el índice escogido para acceder a la tabla.
 - **ref**: recoge las columnas empleadas para buscar valores en el índice.
 - **rows**: devuelve una estimación (basada en estadísticas) de las filas que MySQL necesitará examinar para encontrar los valores buscados. Una aproximación al número total será el producto de los valores mostrados para todas las tablas en esta columna.
 - **partitions**: únicamente aparece cuando se emplea **EXPLAIN PARTITIONS**, e indica las particiones que contienen filas que cumplen las condiciones de búsqueda.
 - **extra**: recoge información adicional, como la necesidad de utilizar tablas temporales, buffers, o *filesort* para ordenar los registros devueltos, en contraposición a utilizar un índice para ordenar.

Como muestra de la utilidad de la sentencia **EXPLAIN**, las tablas 2.4 y 2.5 muestran los resultados de ésta para una consulta antes y después de optimizar índices (reduciendo el tiempo en un 90 %).

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	rmc	ALL	PRIMARY				16251	Using where
1	PRIMARY	<derived2>	ALL					15553	
1	PRIMARY	<derived4>	ALL					15539	
1	PRIMARY	fo	ref	UNIQ_6EDB5	UNIQ_6EDB5	5	fcd.id	1	
1	PRIMARY	bdd	eq_ref	PRIMARY	PRIMARY	4	rm_walva.fo.bank_details_id	1	
1	PRIMARY	<derived5>	ALL					14816	
1	PRIMARY	<derived6>	ALL					1	
1	PRIMARY	<derived7>	ALL					12876	

Tabla 2.4: EXPLAIN antes de optimizar consulta. Duración de más de 5 minutos.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	rmc	range	PRIMARY	PRIMARY	12		16214	Using where
1	SIMPLE	rmcdp	eq_ref	PRIMARY	PRIMARY	12	rm_mvno.rmc.Id_Cliente	1	
1	SIMPLE	fcd	eq_ref	PRIMARY	PRIMARY	152	func	1	
1	SIMPLE	fo	ref	UNIQ_6EDB5	UNIQ_6EDB5	5	rm_bi_nds.fcd.id	1	
1	SIMPLE	bdd	eq_ref	PRIMARY	PRIMARY	4	rm_walva.fo.bank_details_id	1	
1	SIMPLE	srp	ref	PRIMARY	PRIMARY	47	func	159	
1	SIMPLE	rmcn	eq_ref	PRIMARY	PRIMARY	12	rm_mvno.rmc.Id_Cliente	1	
1	SIMPLE	rmm	eq_ref	PRIMARY	PRIMARY	12	rm_mvno.rmc.Id_Cliente	1	

Tabla 2.5: EXPLAIN tras optimizar consulta. Se crearon índices y tablas temporales. Duración de unos 45s.

3

Introducción a *Data Warehousing*

Uno de los activos más importantes de cualquier organización es la información de la que dispone. Entre los sistemas que almacenan y manejan estos datos, se distinguen los sistemas operacionales, y los analíticos.

Los sistemas operacionales o transaccionales, también denominados OLTP (*On-Line Transaction Processing*) se encargan de capturar y almacenar transacciones individuales. Su objetivo es poder manejar en tiempo real gran cantidad de flujos de información relativamente pequeños de la forma más rápida y efectiva posible. Son la herramienta a emplear para realizar las tareas rutinarias de la organización, y no suelen almacenar demasiados datos históricos.

En contraposición a éstos, los sistemas analíticos, conocidos como OLAP (*On-Line Analytical Processing*), dan soporte al análisis de grandes cantidades de información histórica, alimentada a partir de los sistemas operacionales con cierta frecuencia. El número de consultas al que se ve sometido el sistema es mucho menor que un sistema transaccional, pero la complejidad de éstas será bastante mayor, ya que los usuarios querrán analizar información resumida a partir de miles de registros. [14]

En este capítulo se estudiarán las peculiaridades de un sistema OLAP, como es el caso de un *Data Warehouse* (DWH).

3.1. Definición y funciones

“Un *Data warehouse* es una colección de datos orientada a temas, integrada, no volátil y variante en el tiempo que apoya la toma de decisiones de la dirección.”

– Bill Inmon

El padre del *Data Warehousing* definía así, en 1996, uno de los conceptos más importantes en *Business Intelligence*. [15]

Una organización suele tener varios sistemas operacionales. Cada uno de ellos genera información siguiendo sus propias normas, métricas y convenciones. Para que la dirección de diversos departamentos pueda tomar decisiones informadas, es necesario *integrar* la información de estas fuentes, de forma que sea posible obtener una visión global del área de negocio a analizar.

La información de los sistemas transaccionales es volátil. Los registros pueden ser insertados, actualizados y eliminados. Sin embargo, uno de los objetivos de un *data warehouse* es mantener información histórica. Mediante el principio de *no volatilidad*, se establece que los registros no sean eliminados una vez han entrado al repositorio. En cuanto a las actualizaciones, interesa mantener la trazabilidad de los cambios que se hagan en las fuentes sin afectar a los registros ya existentes en el *data warehouse*. Gracias a esto, es posible recuperar una fotografía (*snapshot*) del sistema a fecha pasada, siendo el sistema *variante en el tiempo*.

Uno de los aspectos más importantes a tener en cuenta a la hora de diseñar un repositorio es que su verdadero fin es ayudar a los usuarios, normalmente directivos y gerentes, a tomar decisiones de negocio informadas. Cualquier negocio tiene diversos procesos que pueden ser observados de forma independiente, según su *tema*.

Ralph Kimball[16], el otro gran nombre en *data warehouse*, defiende explícitamente que esos *temas* sean procesos de negocio. Para él, la información presentada en el sistema debe ser creíble, intuitiva y de fácil acceso e interpretación para el usuario.

Además, insiste en la necesidad de que el modelo escogido sea adaptativo, extensible y resistente en un entorno tan volátil como es el de los negocios. La verdadera medida de calidad de un sistema de este tipo no va a ser su elegancia, sino la aceptación de los usuarios, que deben ver probada la capacidad del sistema para ayudarles a tomar decisiones.

3.2. Arquitectura de un Data Warehouse

Conceptualmente, el esquema básico de un sistema de DWH consta de tres capas, mostradas en la figura 3.1. A continuación se estudiarán las características y funciones de cada una.

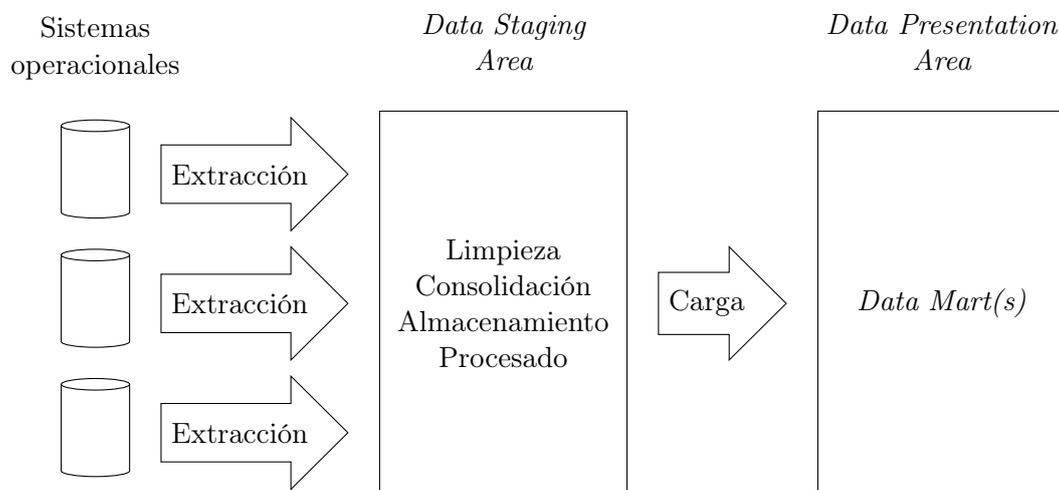


Figura 3.1: Esquema básico de un *data warehouse*.

3.2.1. Sistemas operacionales

Los sistemas operacionales registran las transacciones del negocio. En general, al ser externos al *data warehouse* no tenemos control sobre ellos. Únicamente podemos realizar consultas contra ellos, y no podemos asumir que contengan información histórica consolidada entre ellos. Es más, puede haber descuadres entre los datos proporcionados por las distintas fuentes.

3.2.2. Data Staging Area

El DSA (*Data Staging Area*) es la parte del DWH a la cuál el usuario no puede acceder. Situada entre los sistemas operacionales y el área de presentación, engloba los procesos comúnmente denominados ETL (*Extract-Transformation-Load*) o ELT (*Extract-Load-Transformation*).

ETL y ELT

Una ELT y una ETL tienen en realidad las mismas funciones, aunque el orden en el que las realizan es distinto:

- **Extracción:** lectura e interpretación de los datos de los sistemas transaccionales, y copia al DSA para su manipulación posterior.
- **Transformación:** limpieza y consolidación de los datos. Entre los procesos comunes podemos encontrar la corrección de errores tipográficos, la resolución de conflictos, la estandarización de formatos, la combinación de datos de diferentes fuentes y la detección y eliminación de registros duplicados.
- **Carga:** copia masiva de información a los *data marts*.

Si bien en general se ha utilizado ETL –y es el concepto que desarrolla la mayor parte de la literatura–, cada vez más desarrolladores están comenzando a utilizar ELT. Aunque el tiempo de desarrollo suele ser menor al utilizar ETL, los proyectos de este tipo son menos flexibles, conllevan más riesgo y son más difíciles de paralelizar.[17]

Dado que el desarrollo de la ETL es posiblemente la tarea más compleja en un proyecto de DWH, algunos autores están realizando esfuerzos por definir un método estándar. [18]

Teniendo estos objetivos en mente, podemos entrar con más detalle en la arquitectura del *Data Staging Area* propuesta por Rainardi [19], mostrada en la figura 3.2.

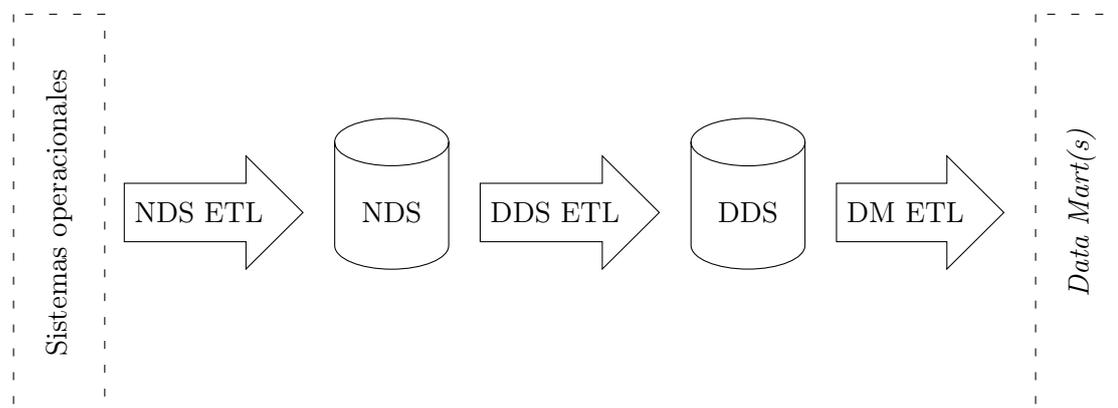


Figura 3.2: Esquema básico de un *Data Staging Area*.

3.2.2.1. NDS (*Normalized Data Store*)

El primer paso a realizar es integrar y guardar históricos de los datos de los diversos sistemas en una única base de datos. Con el fin de minimizar la redundancia, se recurre a una base de datos normalizada. El nivel de normalización se alcanza mediante el cumplimiento de ciertos requisitos:

- Primera forma normal (1NF)
 - No debe haber columnas o grupos de columnas repetidos.
 - Cada tabla debe tener una clave primaria que identifique de forma unívoca cada fila.
- Segunda forma normal (2NF)
 - Cumple 1NF.
 - Cuando el valor de una columna (`columna1`) está asociado al de otra columna (`columna2`), decimos que `columna1` es dependiente de `columna2`. Por ejemplo, el nombre de un cliente depende del identificador de cliente. Para cumplir 2NF, todas las columnas de una tabla deben depender de toda la clave primaria (no solo de una de las columnas que la componen). De no ser así, debe dividirse la tabla en sucesivas tablas que cumplan esta condición.
- Tercera forma normal (3NF)
 - Cumple 2NF.
 - Cuando el valor de una columna (`columna1`) depende del de otra columna (`columna2`), y ésta es a su vez dependiente de una tercera (`columna3`), decimos que `columna3` es transitivamente dependiente de `columna1`. Esta dependencia transitiva no está permitida en 3NF. La tabla debe dividirse.

Aunque existen formas normales de mayor orden, en la práctica se suele usar la tercera forma normal. Así, se garantiza una base de datos estructurada y no redundante, a costa de una mayor complejidad de las consultas.

3.2.2.2. DDS (*Dimensional Data Store*)

La tercera forma normal es de gran ayuda a la hora de procesar datos operacionales. Sin embargo, un modelo de datos normalizado resulta demasiado complejo para las consultas a un DWH. No es trivial para un usuario el entender, recordar o navegar por un sistema normalizado. De hecho, manejar tales sistemas tampoco es sencillo para un RDBMS.

Como alternativa ante estos problemas aparecen los modelos dimensionales, cuyo objetivo es ofrecer la misma información almacenada en un NDS poniendo énfasis en su inteligibilidad, rendimiento y resiliencia al cambio, a costa de sacrificar espacio en disco.

Cuando, como en nuestro caso, la tecnología subyacente es una base de datos relacional, el modelo recomendado por Kimball es el *star schema* (modelo de datos en estrella), mostrado en la figura 3.3. Cada estrella está compuesta por una tabla de hechos y entre 5 y 15 dimensiones.

La mejor forma de comprender los bloques básicos de un sistema dimensional es mediante un ejemplo. Supongamos el siguiente requerimiento:

“Necesito saber el número de ítems vendido por día de cada producto.”

– El CEO de cualquier cadena de supermercados

La tabla de hechos (*fact table*) es la tabla principal en un modelo dimensional. Además de definir una relación N:M al conectar con las claves primarias de cada una de las dimensiones (manteniendo integridad referencial), almacena una serie de medidas tomadas del proceso de

negocio en análisis, así como cálculos realizados a partir de éstas. Estos datos normalmente son numéricos y aditivos, como en nuestro ejemplo la cantidad de los artículos vendidos.

El concepto más importante a la hora de diseñar una tabla de hechos es el de granularidad (*grain*), que define el nivel de detalle de ésta. Es decir, al especificar la granularidad se está fijando qué hecho debe describir una fila en la tabla. Siguiendo el ejemplo anterior, podríamos tener una fila para cada transacción y producto. Estas columnas formarán la Clave Primaria de la tabla. Hay que tener en cuenta que solo debe generarse una fila en una tabla de hechos cada vez que *ocurra* algo, de forma que no saturamos la tabla con filas llenas de valores nulos.

Normalmente, las tablas de hechos suelen tener un gran número de filas –llegando a aglutinar el 90 % del espacio consumido por el DWH–, pero con un número relativamente pequeño de columnas.

En el requerimiento recibido, por otra parte, las dimensiones vienen especificadas tras la palabra *por*. Contienen un número discreto de atributos reales a través de los que va a entrar el usuario. En nuestro caso, la dimensión *Producto* contendría todos los datos acerca de cada uno de los artículos que la organización tiene a la venta: su nombre, categoría, marca, distribuidor, embalaje, etc; mientras que la dimensión *Fecha* contendrá todas las fechas de un rango necesario con atributos que faciliten la interpretación: diversos formatos, el día de la semana, el mes y el año, si es laborable o festivo, etc. Además, para evitar que la tabla de hechos tenga valores nulos, se suele añadir una fila extra a la que apuntar los valores que falten o no apliquen.

Aunque las tablas contengan información redundante, este tipo de modelo permite al usuario ver con la máxima facilidad datos agrupados por día, semana, mes... o por cada producto, fabricante, o categoría. De nuevo, nos interesará mantener la máxima granularidad posible, de forma que el análisis pueda ser tan detallado como sea necesario.

Por otra parte, el sistema resulta fácilmente extensible: el impacto de incluir un nuevo atributo o medida en el modelo se limita a crear una columna en la tabla correspondiente, mientras que añadir una nueva dimensión solo supone la creación de una nueva tabla, y la adición de una nueva columna a la tabla de hechos.

Ahora que los conceptos básicos del modelo están claros, podemos pasar a examinar algunos conceptos más avanzados que pueden resultar de utilidad. Un ejemplo son las tablas de hechos sin hechos (*factless fact table*), que pueden ser necesarias cuando únicamente se desea capturar la relación entre las dimensiones.

En otros casos, podemos encontrarnos una dimensión degenerada (*degenerate dimension*) cuando queremos mantener identificadores como el número de pedido. Este identificador daría lugar a una dimensión vacía –puede interesarnos poder ver el importe total de cada pedido–, por lo que simplemente se almacena en la tabla de hechos, aun no apuntando a ninguna tabla de dimensión.

Aunque Kimball recomienda evitar normalizar el DDS, puede ser útil unir una nueva dimensión a otra ya existente cuando se busque añadir información adicional a la dimensión y la granularidad difiera en gran medida. Este tipo de relación se denomina *outrigger*. La figura 3.4 muestra un ejemplo en el que se ha añadido una dimensión con información sobre la demografía de una provincia, dato que está incluido en la dimensión *Cliente*. En este caso el modelo pasaría de ser en estrella, a ser un modelo en copo de nieve (*snowflake schema*).

Por último, al tratar con sistemas operacionales complejos podemos encontrarnos con multitud de *flags* asociados a cada registro, pudiendo tomar cada uno de ellos un número relativamente pequeño de valores. En ese caso, especialmente cuando hay cierta correlación entre los atributos, el camino a seguir suele ser agruparlos en una *junk dimension*, como se denomina una tabla que contiene varias dimensiones.

3.2.3. *Data Presentation Area (DPA)*

El área de presentación de datos es el lugar en el que los datos, tras su organización, quedan almacenados para la consulta y el análisis. Al hablar del DPA, en general nos estamos refiriendo a un conjunto de *Data Marts* (DM). De acuerdo con la metodología Kimball, cada uno de ellos debe presentar los datos de un único proceso de negocio. Como si de un DDS se tratara, utilizan diseño dimensional, siempre con el objetivo de que el usuario pueda realizar consultas eficientes de forma intuitiva.

Todos los *data marts* deben compartir dimensiones comunes, denominadas *conformed dimensions*. Esto quiere decir que las claves, atributos y descripciones deben ser los mismos.

3.3. Metodologías de desarrollo en *Data Warehouse*

En esta sección se analizarán algunas peculiaridades de gestionar un proyecto de *data warehouse*.

3.3.1. Metodologías *top-down* y *bottom-up*

Pese a que Bill Inmon y Ralph Kimball, las dos figuras más reconocidas en DWH, comparten gran cantidad de conceptos y métodos, difieren en una parte fundamental del planteamiento de un proyecto de este tipo. [15]

La metodología defendida por Inmon se conoce como *top-down approach*. Según ésta, un *data warehouse* debe ser un repositorio centralizado que recoja la información de toda la organización, materializado en una base de datos normalizada con el grano más fino posible. Una vez creado, este repositorio alimentaría tantos *data marts* dependientes como fuera necesario.

Por su parte, el enfoque de Kimball se denomina *bottom-up approach*. Para él, un *data warehouse* es un conjunto de *data marts* que comparten dimensiones conformadas. En este caso, son los DM los que se crean en primer lugar, uno a uno, de forma que al unirlos a través de las dimensiones comunes sea posible obtener una visión global de la organización.

La visión de Inmon ofrece un sistema integrado y centralizado, con menor redundancia, pero de desarrollo más lento y arriesgado.

El método de Kimball puede producir resultado más rápidos –es posible priorizar los DM más importantes–, pero la metodología Inmon puede dar también buenos resultados, especialmente al ser desarrollado de forma iterativa (*diseño → desarrollo → test → despliegue → diseño...*).

Ponniah propone un compromiso entre ambos modelos. En primer lugar, deben tomarse requisitos a nivel corporativo. Después, debe crearse toda la arquitectura del *data warehouse*, para pasar a conformar y estandarizar los datos que contendrá. Por último, tratando de utilizar dimensiones conformadas, deben crearse *supermarkets*, denominación que da a una serie de *datamarts* cuidadosamente diseñados.[15]

3.3.2. Toma de requisitos

A lo largo de todo el capítulo, se ha puesto especial énfasis en que una solución DWH carece de valor si no es aceptada por parte del usuario. Para lograr ajustar el desarrollo a sus necesidades y cumplir sus expectativas, debe ponerse especial cuidado en la fase de toma de requisitos.

Debido a los distintos enfoques que pueden tener desarrolladores y usuarios, es importante encontrar un método de comunicación efectiva, donde los procesos y requisitos funcionales y

de negocio puedan ser definidos fácilmente. Para esto puede merecer la pena ir a un nivel más alto, alejándose del plano técnico, y pasando a un modelo conceptual como los paquetes de información, como se ve en la figura 3.5.

Los paquetes de información (*information package*) son una herramienta muy valiosa para facilitar el entendimiento entre el personal de IT y el usuario. Como se ve en la tabla 3.1, contienen los hechos a medir, las dimensiones a través de las que se va a analizar, y los atributos definidos para cada una de éstas, a ser posible ordenados jerárquicamente. [20]

A priori, los usuarios no tienen por qué saber dar detalles. Sin embargo, como expertos en su campo, saben cómo medir el éxito de su departamento, y cómo desglosar esas medidas para obtener la información que necesitan. Esa es la información que buscamos. [15]

Fecha	Producto	Cliente
Año	Categoría	Edad
Mes	Marca	Género
Semana	Modelo	Ingresos anuales
Flag festivo	Embalaje	Estado civil
	Color	Vehículos propios
Hechos a medir: artículos vendidos, importe total.		

Tabla 3.1: Ejemplo de paquete de información. Cada columna se corresponde con una dimensión, conteniendo éstas los atributos que la forman –a ser posible, ordenados jerárquicamente.

Ponniah propone tomar requisitos mediante una serie de entrevistas, teniendo en mente ciertos objetivos para esta fase. Entre ellos, destacan:

- Definir asuntos comunes entre usuario e IT.
- Diseñar las medidas principales del negocio.
- Decidir cómo los usuarios van a agregar datos.
- Establecer la granularidad.
- Determinar la frecuencia de refresco de información.

Dependiendo de la disposición del usuario a realizar la entrevista y proporcionarnos la información necesaria, propone tres estructuras comunicativas:

- **Pirámide:** es un método inductivo especialmente útil cuando el usuario necesita *calentamiento*. Se comienza con preguntas cerradas, para terminar obteniendo una visión general mediante preguntas abiertas hacia el final de la entrevista.
- **Embudo:** es un procedimiento deductivo, útil con usuarios susceptibles acerca del tema a tratar, o cuando los detalles se necesitan al final. Se comienza con preguntas abiertas, acabando la entrevista con preguntas cerradas.
- **Diamante:** en general es la de mejor resultado. Se rompe el hielo con preguntas cerradas, pasando a obtener una visión general con preguntas abiertas, y perfilando los detalles con preguntas cerradas.

Además de las entrevistas con los usuarios, también es necesaria la comunicación con el equipo técnico de la organización, ya que serán ellos los que puedan proveer información sobre los sistemas operacionales, el nivel de detalle de la información existente en éstos y los informes que se están generando en la actualidad.

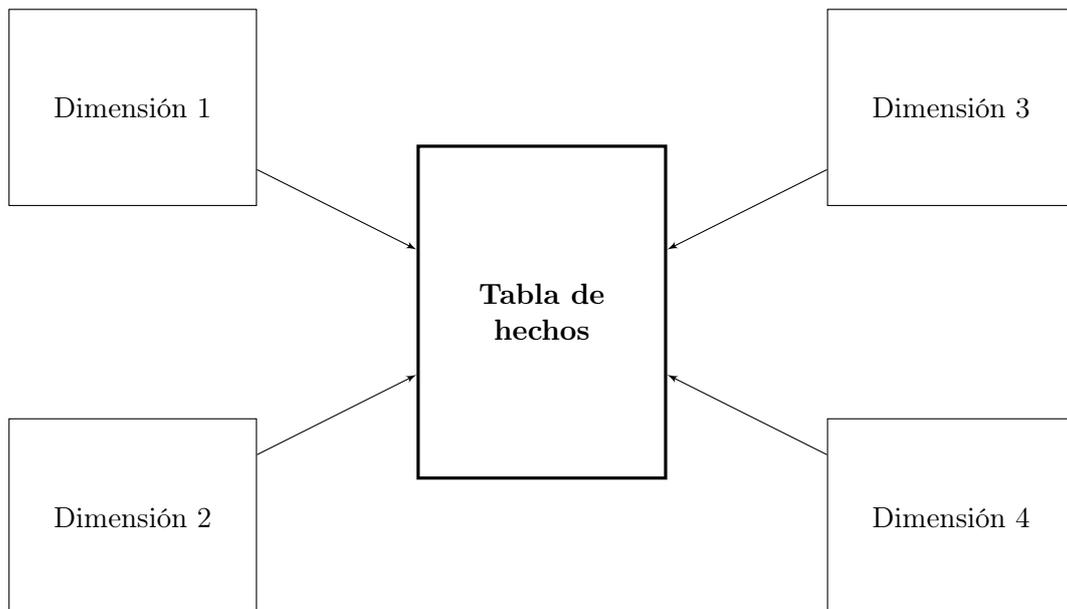


Figura 3.3: Esquema básico de un *star schema*.

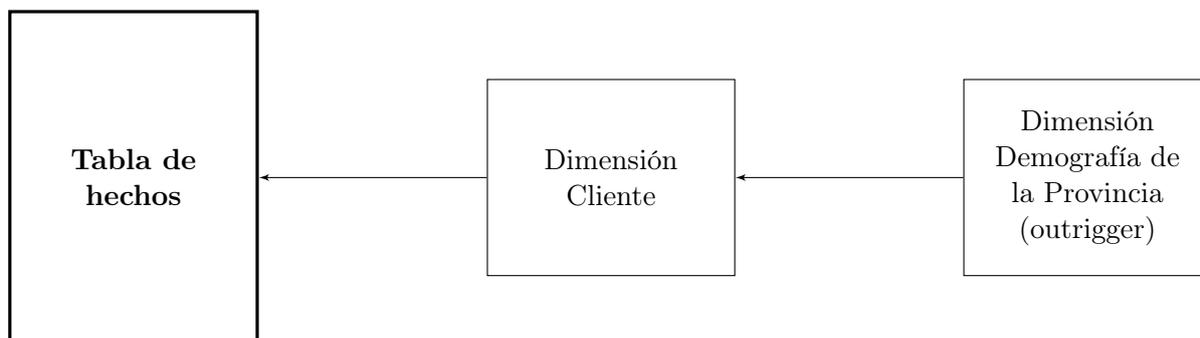


Figura 3.4: Esquema básico de un *outrigger*.

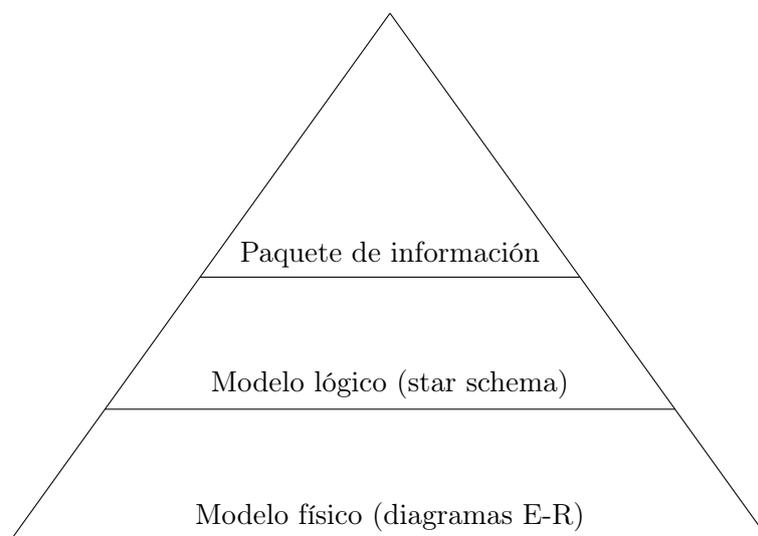


Figura 3.5: Jerarquía de modelos.

4

Introducción a *Data Mining*

Desde el comienzo de la vida humana, la búsqueda de patrones ha sido una tarea habitual. La observación de patrones migratorios a la hora de cazar, de patrones de crecimiento a la hora de cultivar, o incluso de patrones de comportamiento a la hora de relacionarse, así como su análisis de cara a tratar de predecir hechos futuros, han ayudado a nuestra especie a adaptarse a su entorno y, por tanto, a sobrevivir.

Sin embargo, las técnicas y aplicaciones han cambiado a lo largo del tiempo: hemos pasando de la mera observación a gigantescas bases de datos como fuente de información, y de preguntas como *¿va a intentar comerme ese oso?* a otras como *¿qué tipo de productos suelen comprar a la vez mis clientes?* o *¿cuáles de éstas transacciones son fraudulentas?*.

4.1. Definición y aplicaciones

Se comienza a hablar de *data mining* en el momento en el que, mediante técnicas automáticas, se trata de descubrir información valiosa en bases de datos. Asimismo, se pueden utilizar estos métodos para tratar de predecir datos futuros a partir de muestras pasadas. [21][22]

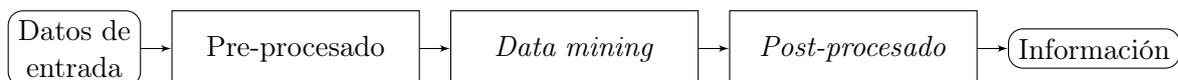


Figura 4.1: Esquema de descubrimiento de conocimiento en bases de datos [22]

Las técnicas de *data mining* son una parte de los esquemas de *knowledge discovery in databases* (KDD), como el mostrado en la figura 4.1. El bloque de pre-procesado se refiere a la extracción de *features* o características, reducción de dimensionalidad, normalización y limpieza de datos en general, mientras que el bloque de post-procesado incluye tareas tales como la visualización, interpretación y filtrado de patrones.

Los problemas a resolver mediante el uso de este tipo de sistemas pueden englobarse en dos grandes categorías: tareas descriptivas, y tareas predictivas.

En el primero de los casos, se busca hallar patrones en los datos: correlaciones, tendencias, *clusters* o agrupamientos, trayectorias y anomalías.

En el caso de las tareas predictivas, el objetivo es estimar el valor de un atributo en concreto (variable dependiente, también denominada etiqueta) basándose en los valores de otros atributos (variables independientes).

Dicho de un modo más formal, para la muestra i -ésima, podemos definir los atributos de entrada como $x^{(i)}$ y la variable que tratamos de predecir como $y^{(i)}$. Hablamos de aprendizaje supervisado cuando contamos con un *dataset* de entrenamiento formado por un m tuplas $\{(x^{(i)}, y^{(i)}); i = 1 \dots m\}$. Si, por el contrario, no contamos con etiquetas en nuestros datos de entrenamiento, hablamos de aprendizaje no supervisado. [23]

En el caso concreto de los problemas de clasificación, el objetivo es predecir una etiqueta $y^{(i)}$ para cada conjunto de atributos de entrada $x^{(i)}$.

4.2. Preparación de datos

A pesar de los avances en el campo del *data mining*, se podría decir que el fenómeno GIGO (*garbage in, garbage out*) tan frecuente en informática también se produce en este caso. En otras palabras: cualquier modelo, por sofisticado que sea, dará resultados erróneos si se entrena a partir de datos inadecuados. Por este motivo, la preparación de datos es imprescindible para posibilitar la creación más rápida de mejores modelos. [24]

De acuerdo con Pyle [24], este proceso puede dividirse en tres pasos:

- **Descubrimiento de datos:** localización y acceso a los datos a emplear.
- **Caracterización de datos:** descripción y comprensión de los datos. En esta fase se decide si la información disponible es útil, veraz y fiable.
- **Ensamblado del *dataset*:** construcción de una representación estándar de los datos entrantes, de forma que puedan ser analizados por una herramienta de modelización.

Si bien cada *dataset* es único, algunos de los problemas que presentan suelen ser comunes. Aunque en algunos casos puede ser necesaria la toma de decisiones de diseño, muchos de éstos problemas tienen soluciones ya conocidas.

A continuación, se analizarán los retos más importantes que presentan los datos a la hora de alimentar un modelo.

4.2.1. Inconsistencias

En la mayoría de los casos resulta necesario reunir información de distintos sistemas. Cada uno de ellos puede tener diferentes criterios a la hora de almacenar y clasificar los datos, provocando descuadres y haciendo más compleja la fase de integración.

Una posible solución ante este tipo de problemas es la utilización de técnicas de Data Warehouse, como las descritas en el capítulo 3.

4.2.2. *Outliers*

Los *outliers* se definen como objetos que, de algún modo, tienen características distintas de la mayoría de los otros objetos del *data set*. También se utiliza este término para denominar aquellos ítems que recogen en un atributo un valor anormal, lejano a los valores típicos que suele tomar esa característica.

Su origen puede ser o no legítimo: pueden aparecer debido a errores de medición o de cálculo, o pueden simplemente ser valores reales inusuales que de hecho estamos tratando de encontrar. Por ejemplo, el objetivo de un sistema de detección de transacciones fraudulentas puede pasar precisamente por detectar eventos sospechosos. [22]

De acuerdo con [25], en la fase de pre-procesado, los *outliers* pueden ser detectados a través de una serie de técnicas de análisis recogidas bajo el nombre de *descriptive data summarization*. Entre las medidas más útiles, se encuentran las siguientes:

Media y mediana

La media aritmética resulta útil para medir la tendencia central de una serie de valores. A partir de N valores x_1, x_2, \dots, x_N , se define matemáticamente como:

$$\bar{x} = \frac{\sum_{i=1}^N x_i}{N} \quad (4.1)$$

En caso de contar con un peso w_i para cada valor x_i , se puede definir la media ponderada como:

$$\bar{x} = \frac{\sum_{i=1}^N w_i x_i}{N} \quad (4.2)$$

El problema de la media es su sensibilidad a valores extremos. En ocasiones, para resolverlo, se descarta un cierto porcentaje de valores mínimos y máximos de cara a su cálculo.

En el caso de conjuntos de datos asimétricos, es común utilizar la mediana como medida de la tendencia central, definiéndose ésta como el valor medio del conjunto ordenado (o la media entre los dos centrales, si el número es impar).

Dispersión

La medida más sencilla de dispersión es el rango:

$$\text{rango}(x) = \max(x_i) - \min(x_i) \quad (4.3)$$

A la hora de detectar *outliers*, sin embargo, los percentiles resultan de mayor utilidad. Asumiendo un conjunto de valores ordenado de menor a mayor, el percentil k -ésimo tendrá el valor x_i tal que el $k\%$ de los valores tengan valor igual o menor a x_i .

Los percentiles 25, 50 y 75 también se conocen como los cuartiles Q_1 , Q_2 y Q_3 . Una medida interesante derivada de ellos es el rango intercuartil (IQR), definido como:

$$IQR = Q_3 - Q_1 \quad (4.4)$$

Una regla práctica para identificar posibles *outliers* es localizar valores por debajo de $Q_1 - 1,5 \times IQR$ o por encima de $Q_3 + 1,5 \times IQR$.

4.2.3. Datos incompletos

Otro de los retos que comúnmente presenta un *dataset* es la existencia de datos incompletos. Pueden aparecer por error, o tener sentido (por ejemplo, el identificador del carnet de conducir de una persona que no conduzca será nulo). De acuerdo con [26], los valores nulos pueden clasificarse en los siguientes grupos:

- MCAR (*Missing Completely At Random*), cuando la distribución de las muestras que contienen un valor nulo para un atributo no dependa de los datos observados ni del valor que falta.
- MAR (*Missing At Random*), cuando la distribución de las muestras que contienen un valor nulo para un atributo dependa de los datos observados, pero no del valor que falta.
- NMAR (*Not Missing At Random*), cuando la distribución de las muestras que contienen un valor nulo para un atributo dependa del valor que falta.

Entre la gran cantidad de metodologías existentes a la hora de manejar datos incompletos, destacan las siguientes:

- Rellenar manualmente los valores: es, por supuesto, el método que mayor probabilidad de acierto garantiza, pero resulta imposible de realizar especialmente para grandes bases de datos.
- Descartar los ítems que contengan algún valor nulo.
- Rellenar con una constante global: es posible asignar una etiqueta “?” a aquellos valores nulos. Esto puede condicionar a la herramienta de modelización, ya que no todas están preparadas para soportar datos incompletos y podrían interpretarlo como una categoría relevante por sí misma. [25]
- Utilizar técnicas estadísticas para tratar de estimar el valor real. Entre ellas, destacan:
 - MC: rellenar con la media o la mediana si el valor es numérico, o con la moda si el valor es nominal. [26]
 - CMC: rellenar con la media o la mediana si el valor es numérico, o con la moda si el valor es nominal, obteniendo estas estadísticas únicamente de ítems etiquetados con la misma clase. [26]
 - KNNI (*K-Nearest Neighbour Imputation*): rellenar con la media o la moda de los valores más cercanos en el espacio de atributos. [26]
 - Emplear regresión lineal para atributos numéricos, o regresión logística para atributos nominales. [25]
 - En el caso de atributos numéricos, recurrir a *Expectation-Maximization* (EM), un algoritmo iterativo que consta de un paso E, durante el cuál se estiman los valores a partir de los datos existentes, y el paso M, en el que se maximiza la función de verosimilitud. [27]

En la literatura hay diversos estudios sobre la efectividad de cada uno de estos métodos a la hora de mejorar el desempeño de un clasificador, si bien depende del *dataset* concreto y el ratio de valores nulos. Su et al. [28] encuentran que la utilización de MC no suele mejorar el clasificador original, mientras que EM tiende a ser la mejor opción especialmente al aumentar el ratio de valores nulos.

Por su parte, el estudio de Luengo et al. [26] concluye que cualquier método que impute valores da mejores resultados que dejar los valores nulos o descartar los ítems que contengan alguno. Por otro lado, encuentran que el desempeño del clasificador depende en gran medida del modelo empleado. CMC es uno de los modelos que menor ruido introduce al rellenar datos. Entre los métodos disponibles en WEKA, el que mejor resultados presenta para clasificadores SVM (descritos en la sección 4.3) es MC, seguido por EM.

4.2.4. Excesiva granularidad

Contar con datos muy detallados puede conllevar tiempos de cálculo demasiado largos, haciendo el análisis poco práctico o, incluso, inviable. Las técnicas de reducción de datos aparecen como respuesta a este reto, ofreciendo métodos para obtener una representación reducida del *dataset* que produzca prácticamente los mismos resultados. Han [25] recoge cinco estrategias:

- Discretizar los atributos numéricos, reemplazando valores brutos por rangos o conceptos.
- Agregar los datos para reducir la granularidad.
- Seleccionar únicamente los atributos más relevantes.
- Disminuir la dimensionalidad mediante técnicas de codificación.
- Emplear muestreo, *clustering* o histogramas para obtener representaciones más pequeñas.

Si bien el objetivo principal era reducir el tiempo de cálculo, se ha demostrado en otros casos que aplicar discretización puede mejorar el desempeño de un clasificador cuando se consigue una representación más consistente y significativa, por lo que profundizaremos en esta opción.

Al igual que las herramientas de modelización, los métodos de discretización pueden dividirse en supervisados y no supervisados, según utilicen o no la etiqueta de clase para inferir los valores frontera entre los rangos generados. La ventaja de los métodos supervisados es su capacidad de encontrar fronteras significativas que ayuden posteriormente a la hora de predecir la clase. En un sentido estricto, sólo los datos de entrenamiento deberían ser empleados para hallar las fronteras, dado que emplear las etiquetas de todo el *dataset* implicaría *ver el futuro*.

Los métodos no supervisados más relevantes [29] son los siguientes:

- EW (*Equal Width*): dividir en rangos del mismo tamaño.
- EF (*Equal Frequency*): distribuir en rangos de tamaño no uniforme de forma que cada uno contenga el mismo número de valores.
- *Clustering* mediante el algoritmo *k-means*.

Por su parte, entre los métodos supervisados, destacan:

- MDL (*Minimum Description Length*): sugerido por Fayyad e Irani [30] y que consiste en encontrar de forma recursiva hasta cumplir un criterio un valor del atributo que minimice la entropía como frontera, definiéndose ésta como:

$$Entropy(D_k) = \sum_{i=1}^m p_i \log_2(p_i)$$

siendo p_i la probabilidad de que el conjunto de tuplas D_k pertenezca a la clase C_i entre m clases. Teniendo en cuenta que el conjunto de tuplas D_1 correspondería a las instancias

que quedaran por debajo de la frontera de decisión, mientras que D_2 contendría aquellas que permanecieran por encima de ésta, se trataría de minimizar la entropía de ambas. Kononenko, más tarde, sugiere emplear este mismo algoritmo siguiendo un criterio distinto de parada. [31]

- *ChiMerge*: basándose en la premisa de que un mismo intervalo debería ser consistente en cuanto a la clase de las instancias que muestran ese valor, se comienza asumiendo que cada valor se corresponde con un intervalo distinto. Utilizando la medida estadística χ^2 , se encuentra qué intervalos adyacentes pueden unirse dada su independencia de la clase. Este procedimiento se ejecuta recursivamente hasta cumplir ciertos criterios. [32]
- CAIM (*Class Attribute Interdependence Maximization*), que trata de encontrar intervalos que contengan instancias de una única clase, asumiendo que habrá al menos tantos intervalos como clases, no siendo necesario definir el número de rangos para los que encontrar frontera. [33] De acuerdo con Cano et al. [34], el hecho de que CAIM encuentre un número de intervalos cercano al de clases no es una solución óptima para todos los problemas, y argumenta que dado que CAIM tiene en cuenta únicamente la clase con más instancias, el desempeño de este método disminuye en *datasets* no balanceados. Ante eso, sugieren una modificación que denominan ur-CAIM.

En la comparativa de Cano et. Al [34], se ve que el tiempo de cálculo requerido por los métodos no supervisados es el menor, mientras que CAIM es el método que más tarda en calcular. En cuanto a precisión, IEM suele decantarse como el mejor algoritmo para *datasets* balanceados, mientras que ur-CAIM suele dar mejores resultados en *datasets* no balanceados.

Por otro lado, la comparativa de Zhu et al. [35] concluye que, en general, IEM obtiene un F_1 score ligeramente mejor que IEMV, y bastante mejor que CAIM, aunque en las ocasiones en las que este último da mejores resultados lo hace por mayor diferencia.

4.2.5. Atributos de tipos no admitidos

La mayoría de herramientas de modelización sólo admiten ciertos tipos de datos. Por ejemplo, únicamente se puede aplicar regresión lineal a datos numéricos.

En caso de contar con atributos nominales entre las variables de entrada, es necesario convertirlos antes de alimentar el modelo con ellos. Para resolver este problema se puede recurrir a las *dummy variables*. Si, por ejemplo, contáramos con un atributo `tipo_documento` que tome un valor entre NIF, NIE, CIF o Pasaporte, podría sustituirse por cuatro atributos binarios: `tipo_documento=NIF`, `tipo_documento=NIE`, `tipo_documento=CIF` y `tipo_documento=Pasaporte`.

También puede darse el caso contrario. Algunas herramientas pueden no soportar atributos numéricos. Para resolver ésto, es posible recurrir a la discretización, explicada en la sección 4.2.4.

4.2.6. *Datasets* no balanceados

En muchos problemas reales de clasificación la distribución de clases en el *dataset* no tiene por qué ser uniforme. Por ejemplo, el número de usuarios que incurren en fraude será varias veces menor que el de aquellos que no lo hacen.

La primera opción ante este reto es la de aplicar un mayor peso o coste a las instancias de la clase con menor representación, de forma que el clasificador dé mayor importancia a la clasificación correcta de éstas.

La segunda opción es recurrir a técnicas de muestreo con el objetivo de que el número de instancias de cada clase sea aproximadamente igual. En primer lugar, podemos submuestrear la clase con mayor representación, a costa de disminuir el tamaño del *dataset* y perder ciertas propiedades que podrían ser útiles en el análisis. Estos son los motivos explicados por Akbani et al. [36] a la hora de buscar otro tipo de método.

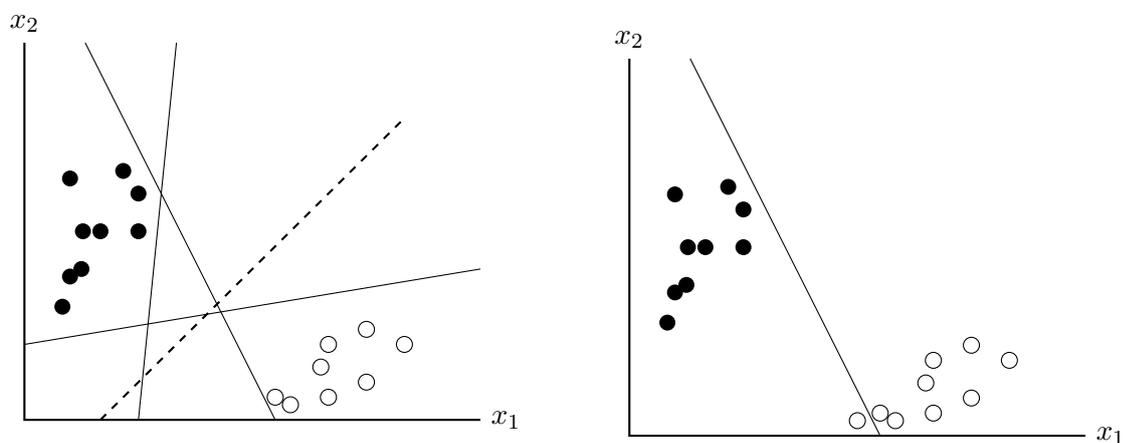
La otra posibilidad, por tanto, es sobremuestrear la clase con menor representación. Un algoritmo muy popular es SMOTE (*Synthetic Minority Over-sampling TEchnique*), descrito por Chawla et al. [37]. Este método consiste en tomar cada instancia de la clase menos representada, escoger n de las k muestras más cercanas en el espacio de características (siendo n y k parámetros del algoritmo, y escogiendo aleatoriamente si $n < k$), e introducir una muestra sintética en el segmento que une la muestra original y sus vecinas.

Experimentalmente, Akbani et al. [36] encontraron que la utilización de SMOTE mejora ampliamente el desempeño de un clasificador SVM. Sin embargo, y aunque el empleo de submuestreo da en algunos casos mejor resultado que SMOTE, la mayor precisión se obtiene combinando SMOTE y costes.

4.3. SVM (*Support Vector Machine*)

Las Máquinas de Soporte Vectorial, más conocidas como SVMs, son una técnica de clasificación propuesta por Vapnik y Chervonenkis en 1963 que se hizo popular al ser aplicada con éxito en problemas de reconocimiento de caracteres manuscritos. SVM da muy buenos resultados al ser empleado sobre datos de alta dimensionalidad, siendo una de sus peculiaridades que únicamente utiliza un subconjunto del *dataset* para representar la frontera de decisión. Estos puntos se denominan vectores soporte.

SVM utiliza únicamente atributos numéricos de entrada. De ser necesario incluir atributos nominales en el modelo, es necesario recurrir a las técnicas explicadas en la sección 4.2.5.



(a) Posibles fronteras entre dos clases linealmente separables. La mejor frontera aparece punteada.

(b) Comportamiento de una frontera incorrecta ante nuevas muestras.

Figura 4.2: Posibles fronteras de decisión

La figura 4.2a¹ es una representación gráfica de muestras pertenecientes a dos clases, denotadas como \bullet y \circ . En este caso, el *dataset* es linealmente separable: es posible encontrar un

¹Las figuras de esta sección están basadas en código L^AT_EX de Yifan Peng, disponible en <http://blog.pengyifan.com/tikz-example-svm-trained-with-samples-from-two-classes/>.

hiperplano que divide las instancias según su etiqueta. Es más, pueden encontrarse infinitas fronteras que cumplen a la perfección este cometido.

Sin embargo, intuitivamente, podemos ver que la frontera punteada es mejor que el resto, dado que está más alejada de ambos grupos. Si, como muestra la figura 4.2b, se escogiera otra frontera, el clasificador sería más vulnerable a nuevas muestras que, aun estando muy cerca de las otras de su clase en el espacio de características, serían incorrectamente clasificadas.

De manera más formal, nos enfrentamos a un problema de clasificación. Cada una de las N instancias del conjunto de entrenamiento viene denotada por la tupla (\mathbf{x}_i, y_i) , donde $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})^T$ es un vector que contiene los d atributos de la instancia i . La clase podrá tomar los valores $y_i = \{-1, 1\}$. La frontera de decisión puede expresarse como:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

siendo \mathbf{w} y b parámetros del modelo, que deben ser elegidos de forma que se cumpla:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_i + b \geq 1 & \text{si } y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 & \text{si } y_i = -1. \end{cases}$$

Estas condiciones pueden expresarse de forma compacta como:

$$\hat{\gamma}_i = y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

Definiendo $\hat{\gamma}_i$ como el margen funcional con respecto a la instancia i , el margen funcional con respecto al *dataset* se podría definir como el peor caso:

$$\hat{\gamma} = \min \hat{\gamma}_i = \min y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$$

Por otro lado, se puede demostrar que la distancia geométrica entre una instancia y la frontera puede expresarse como:

$$\gamma_i = y_i \left(\left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \right)^T \cdot \mathbf{x}_i + \frac{b}{\|\mathbf{w}\|} \right)$$

Del mismo modo que con el margen funcional, el margen geométrico del *dataset* se puede definir como el peor caso:

$$\gamma = \min \gamma_i$$

Para obtener el mejor clasificador posible debemos maximizar el margen funcional. En otras palabras, hemos de obtener los parámetros \mathbf{w} y b tal que cada muestra tenga al menos un margen funcional γ . Nos enfrentamos al siguiente problema de optimización:

$$\min_{\gamma, \mathbf{w}, b} \gamma \quad \text{tal que} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq \gamma, \quad \|\mathbf{w}\| = 1$$

Aprovechando que $\hat{\gamma} = \|\mathbf{w}\| \cdot \gamma$, escalando el problema de forma que $\hat{\gamma} = 1$, y sabiendo que maximizar $\hat{\gamma}/\|\mathbf{w}\|$ es equivalente a minimizar $\|\mathbf{w}\|^2$, el problema de optimización se puede afrontar como:

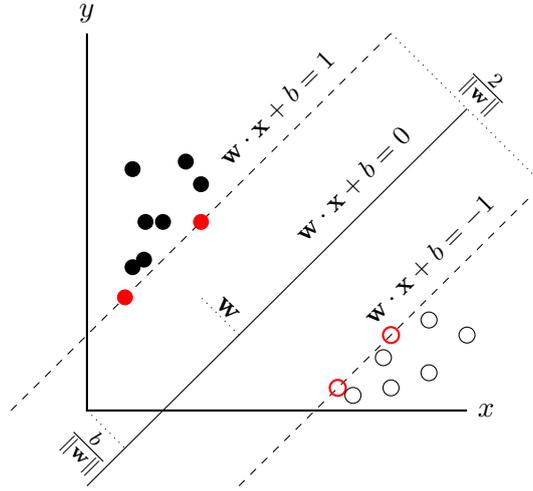


Figura 4.3: Frontera de decisión y margen de un clasificador SVM ante datos separables.

$$\min_{\gamma, \mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{tal que} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

Esta expresión, cuadrática y convexa, puede ser minimizada eficientemente mediante el método de los multiplicadores de Lagrange.

Sin embargo, en *datasets* reales, no tiene por qué ser tan fácil separar los datos. En esta situación, representada en la figura 4.4, SVM se enfrenta a un compromiso entre el tamaño del margen y el número de errores de entrenamiento causados por la frontera lineal. Este enfoque se denomina *soft margin approach*. Como se puede apreciar en la imagen, la frontera B_1 clasifica correctamente todas las instancias, pero tiene un menor margen. Sin embargo, a costa de clasificar incorrectamente dos de las muestras, la frontera B_2 tiene un mayor margen y, por tanto, no se ajustará tanto a los datos de entrenamiento que perderá generalidad. En otras palabras, será menos sensible a *overfitting*.

Matemáticamente, se puede introducir la variable ξ_i , que da una estimación del error cometido para la muestra i :

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_i + b \geq 1 - \xi_i & \text{si } y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 + \xi_i & \text{si } y_i = -1. \end{cases}$$

Teniendo esto en cuenta, y añadiendo el parámetro $C > 0$, correspondiente a la penalización sobre el error, el problema de optimización resultante es el siguiente:

$$\min_{\gamma, \mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i \quad \text{tal que} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

La última asunción que debemos eliminar es la de que los datos pueden ser separados con una frontera lineal. Para hacer frente a esto, SVM transforma la coordenada original en el espacio de \mathbf{X} a un nuevo espacio $\Phi(\mathbf{X})$ en el cual sí puede emplearse una frontera lineal. Si, por ejemplo, se empleara la transformación:

$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1) \quad (4.5)$$

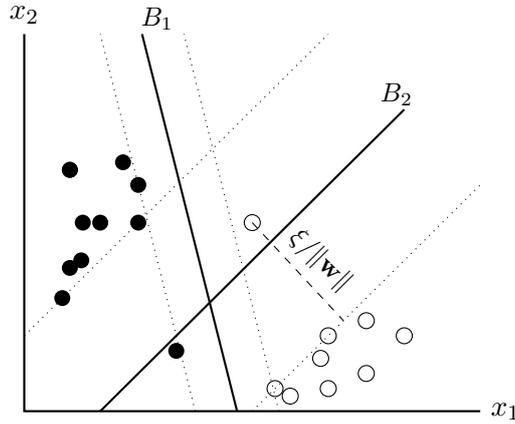


Figura 4.4: Frontera de decisión y margen de un clasificador SVM ante datos no separables.

se estaría aumentando la dimensionalidad del problema, aumentando la probabilidad de caer en la *maldición de la dimensión*: los datos podrían dispersarse demasiado. Para ello, se recurre al denominado *kernel trick*: teniendo en cuenta que el producto escalar entre dos instancias puede interpretarse como una medida de la similitud entre ellas, el producto escalar $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ se puede entender como la similitud de dos instancias en el espacio transformado.

Así, teniendo en cuenta que el principal requisito para que una función pueda ser usada como *kernel* es que cumpla el teorema de Mercer, que garantiza que ésta pueda ser representada como el producto escalar entre dos vectores en un espacio de alta dimensionalidad, se puede definir una función *kernel* como:

$$K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$$

En el caso de la transformación de la ecuación 4.5, al operar, se comprueba que $K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + 1)^2$. Al utilizar un *kernel* estamos consiguiendo calcular la similitud entre dos instancias en el espacio transformado desde el espacio sin transformar. Al tener menor dimensionalidad, además de ahorrar en coste computacional, evitamos los problemas que acarrea la *maldición de la dimensión*.

Los *kernels* más comúnmente usados son:

- Lineal: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \cdot \mathbf{x}_j$
- Polinomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \cdot \mathbf{x}_j + r)^d, \gamma > 0$
- RBF (*Radial Basis Function*): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2), \gamma > 0$
- Sigmoide: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \cdot \mathbf{x}_j + r)$

Teniendo el *kernel* en cuenta, la función a optimizar es la siguiente:

$$\min_{\gamma, \mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i \quad \text{tal que} \quad y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1$$

Mientras que, a la hora de clasificar una instancia de test (\mathbf{z}), se empleará:

$$f(\mathbf{z}) = \text{sign} \left(\sum_{i=1}^m \lambda_i y_i K(\mathbf{x}_i, z) + b \right),$$

siendo λ_i los multiplicadores de Lagrange. [22] [23]

En un sentido más práctico, Hsu et al. [38], autores de `libsvm`, una de las librerías SVM más utilizadas, dan algunos consejos de utilización. De acuerdo con ellos, el *kernel* RBF es en general una primera buena opción, ya que puede manejar relaciones no lineales entre atributos y clases, además de tener menos dificultades numéricas que el *kernel* polinómico.

Para optimizar la elección de los parámetros C y γ , sugieren emplear una búsqueda en malla, probando combinaciones de los parámetros una a una mediante *cross-validation*. Entre los valores que sugieren como prueba inicial están $C = 2^{-5}, 2^{-3}, \dots, 2^{15}$ y $\gamma = 2^{-15}, 2^{-13}, \dots, 2^3$.

4.4. Reglas asociativas

Las reglas asociativas son un método no supervisado diseñado en un primer momento para encontrar relaciones ocultas en grandes *datasets*. El ejemplo más típico de aplicación es el del análisis de cestas de la compra, donde se trata de encontrar patrones interesantes de productos que han sido comprados en una misma transacción.

Las reglas asociativas están diseñadas para trabajar con atributos binarios, por lo que suelen emplearse *dummy variables*, explicadas en la sección 4.2.5. Un ejemplo de tabla con varias transacciones se muestra en la tabla 4.1. Podemos definir $I = \{i_1, i_2, \dots, i_d\}$ como el set de todos los ítems en una transacción, siendo el conjunto de todas ellas $T = \{t_1, t_2, \dots, t_N\}$. Cada transacción t_i , por tanto, contendrá algunos ítems de I .

Transacción	Pizza	Cerveza	Leche	Huevos
1	1	0	0	1
2	0	1	1	1
3	1	1	1	1

Tabla 4.1: Ejemplo de transacciones binarizadas.

Decimos que una transacción t_i contiene un conjunto de ítems X si X es un subconjunto de t_i . El número de transacciones que contiene un conjunto de ítems en particular puede definirse matemáticamente como:

$$\sigma(X) = |\{t_i \mid X \subseteq t_i, t_i \in T\}|.$$

A partir de este valor, podemos calcular las medidas más importantes de la relevancia de una regla asociativa: el soporte y la confianza. Para una regla $X \rightarrow Y$, donde X (la premisa) e Y (la consecuencia) son conjuntos de ítems disjuntos (i.e. $X \cap Y = \emptyset$), pueden definirse como:

$$\text{Soporte}(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

$$\text{Confianza}(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

El soporte mide la proporción en la que se cumple la regla en el *dataset* completo (cuántas muestras del conjunto de datos cumplen $X \rightarrow Y$), mientras que la confianza es un indicador del

cumplimiento de la regla sobre la premisa (cuántas muestras que cumplen X cumplen también $X \rightarrow Y$). Un alto soporte garantiza que una regla no se da por casualidad, mientras que una alta confianza indica una alta probabilidad de que la consecuencia ocurra al darse la premisa. De hecho, la confianza puede interpretarse como una estimación de la probabilidad condicionada de $P(Y | X)$. Los valores mínimos de cada una de estas medidas son, además, parámetros del algoritmo de búsqueda.

Aunque sería posible buscar reglas por fuerza bruta, este procedimiento resulta excesivamente costoso incluso para *datasets* con pocos atributos. Además, suelen emplearse umbrales mínimos de confianza y soporte para eliminar reglas irrelevantes, por lo que buena parte de las reglas que tanto esfuerzo costó encontrar serían tristemente desechadas. Por ello, los algoritmos suelen descomponer este problema en dos tareas: la generación de *itemsets* frecuentes, que localiza todos los conjuntos que superan el umbral de soporte, y la generación de reglas, que extrae reglas que superan el umbral de confianza a partir del conjunto de ítems frecuentes.

Una de las estrategias más utilizadas en el primero de estos pasos es el principio Apriori, que establece que si un *itemset* es frecuente, también han de serlo todos los subconjuntos de éste. Esta regla se cumple también en el caso negativo: todos los subconjuntos de un *itemset* infrecuente lo serán de la misma forma. Esto puede facilitar el descarte de un alto número de conjuntos a evaluar en cuanto encontramos uno que no tenga suficiente soporte.

Utilizando este principio, el algoritmo encuentra conjuntos frecuentes de forma iterativa, aumentando progresivamente el número de elementos de cada uno. Finalmente, buscará reglas asociativas sobre éstos.

4.5. Evaluación de rendimiento

A la hora de comparar clasificadores es necesario estandarizar una serie de métricas que permitan conocer con exactitud su capacidad de acierto en distintas situaciones. Para ello, el conjunto de datos suele ser dividido en conjuntos de entrenamiento y test. Después, utilizando las etiquetas del segundo y las predicciones arrojadas por el modelo, es posible obtener una visión detallada de la capacidad predictiva de éste.

4.5.1. División del *dataset*

El objetivo de los modelos predictivos es aprender, dados una serie de atributos de entrada, a distinguir las clases de las instancias futuras no etiquetadas. Dado que necesitamos evaluar el desempeño del modelo sobre datos a los que no ha sido expuesto en la fase de entrenamiento, se debe fraccionar el *dataset* en distintos conjuntos independientes.

Es por esto que, en general, y tras aleatorizarlo, se crean dos conjuntos a partir del *dataset*: el conjunto de entrenamiento, formado por instancias etiquetadas a partir de las cuales la herramienta aprenderá a clasificar; y el conjunto de *test*, que contendrá las muestras a las que se expondrá el clasificador tras su entrenamiento, comparando la clase predicha con la clase correcta para evaluar el rendimiento del clasificador. Dada la importancia de que el modelo no aprenda del conjunto de test, debe realizarse la división antes de aplicar el pre-procesado, especialmente si este utiliza algún algoritmo supervisado.

En algunos casos, además, se reserva un cierto porcentaje de muestras para formar un conjunto de validación, utilizado para evaluar el impacto de ciertos parámetros utilizados tras el entrenamiento en algunos modelos. [21] Por ejemplo, sería de utilidad para escoger un valor para el umbral de decisión en el caso de emplear regresión logística.

Sin embargo, en otras ocasiones es necesario optimizar ciertos parámetros que se aplican durante el entrenamiento, como C y γ en el caso de un clasificador SVM. En ese caso, se suele recurrir al algoritmo denominado *k-fold cross-validation*, consistente en los siguientes pasos para un dataset S formado por m instancias [23]:

1. Dividir de forma aleatoria las muestras de S en k conjuntos disjuntos S_1, \dots, S_k .
2. Para $j = 1, 2, \dots, k$:
 - a) Entrenar el modelo con el conjunto $S'_j = \{\cup_{i \neq k} S_i\}$ como set de entrenamiento.
 - b) Obtener el error de generalización empleando el modelo sobre S_j .
3. Calcular el error medio como la media de los errores de cada j .

Este método garantiza dejar fuera de la evaluación la menor cantidad posible de datos, y puede modificarse para calcular cualquier métrica. Es común la utilización de $k = 10$, ya que se ha demostrado que da la mejor estimación del error. [21]

4.5.2. Medidas de rendimiento

Cuando nos enfrentamos a un problema de clasificación con dos clases, podemos interpretarlo como la pertenencia o no de cada instancia a una de ellas. Así, se pueden definir los siguientes conceptos:

- TP (Verdaderos positivos), el número de muestras de clase positiva clasificadas correctamente.
- TN (Verdaderos negativos), el número de muestras de clase negativa clasificadas correctamente.
- FP (Falsos positivos, o error de tipo I), el número de muestras de clase negativa clasificadas incorrectamente como positivas.
- FN (Falsos negativos, o error de tipo II), el número de muestras de clase negativa clasificadas incorrectamente como positivas.

Una representación común de estas medidas es la matriz de confusión, mostrada en la tabla 4.2.

		Clase predicha	
		-	+
Clase real	-	TN	FP
	+	FN	TP

Tabla 4.2: Ejemplo de matriz de confusión.

También podemos calcular la proporción de puntos clasificados correctamente. Esta medida se conoce como precisión:

$$Precisión = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.6}$$

Sin embargo, esta medida puede no ser la mejor para evaluar algunos modelos. Suponiendo un *dataset* no balanceado en el que contamos con un 95 % de muestras positivas, el modelo más simple sería predecir que todas las muestras son positivas, obteniendo un 95 % de precisión en un modelo que realmente no tiene en cuenta ningún aspecto de los datos. Para tener un mayor control sobre este aspecto, se definen la sensibilidad y la especificidad:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \quad (4.7)$$

$$\text{Especificidad} = \frac{TN}{TN + FP} \quad (4.8)$$

En la literatura se emplea la media geométrica de estas medidas, denominada *g-means*:

$$g = \sqrt{\text{Sensibilidad} \times \text{Especificidad}} \quad (4.9)$$

También se definen *precision* y *recall*, muy empleadas cuando la detección de una de las clases se considera más importante que la del resto:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4.10)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4.11)$$

Maximizar ambas debe ser el objetivo de cualquier modelo, por lo que se resumen en una única métrica denominada *F₁ score*:

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4.12)$$

5

Sistemas empleados en un Operador Móvil Virtual

República Móvil es un Operador Móvil Virtual *Reseller* fundado en el año 2013. En la actualidad, las cinco tarifas que ofrecen incluyen un bono de datos a través del cuál conectarse a internet. Las tres tarifas más grandes incorporan, además, bonos de voz.

Si bien en sus inicios la compañía operaba únicamente a través de internet y teléfono, actualmente cuenta también con una red de distribución consistente en tiendas físicas desde las que es posible contratar sus servicios.

Entre las promociones disponibles, destaca el *Plan Pioneros*, que hace que cada nuevo usuario reporte ingresos mensuales a su padrino – referenciado en el momento del alta –, y al padrino de éste. Estos ingresos se traducen en descuentos en la factura mensual. En caso de que superen los gastos del mes en curso la diferencia se añade al saldo de una tarjeta monedero, que será enviada en cuanto éste supere el importe de 10 euros.

5.1. Definición y peculiaridades de un MVNO (*Mobile Virtual Network Operator*)

Las grandes inversiones en licencias de espectro, cableado y equipos de red necesarias para fundar un Operador Móvil Real hacen que las barreras de acceso al mercado sean realmente altas. Dado que el espectro electromagnético es un recurso limitado, los reguladores –en el caso de España, la Comisión Nacional del Mercado y la Competencia– obligan a los operadores que disponen de licencias de espectro a proporcionar servicios al por mayor, permitiendo la entrada de nuevas compañías. Estos nuevos operadores se denominan Operadores Móviles Virtuales (OMV, o MVNO (*Mobile Virtual Network Operator*)). [39]



Figura 5.1: Elementos de un servicio de comunicaciones móviles

Un operador móvil convencional estaría a cargo de todos los equipos y procesos implicados, representados en la figura 5.1:

1. el núcleo de red, que interconecta los distintos operadores;
2. la red de acceso, que conecta usuario y operador;
3. la plataforma de aplicación;
4. los sistemas de gestión de suscriptores;
5. los sistemas CRM (*Customer Relationship Manager*) de facturación;
6. y la imagen de marca, junto con los procesos de ventas y distribución.

Partiendo de la base de que ningún OMV dispone de licencia de explotación del espectro electromagnético, pueden definirse varios tipos según su participación dentro del esquema de un operador móvil completo:

- Un OMV completo (*Full MVNO*) abarca la mayor parte del proceso (puntos 2-6). La única diferencia con un operador móvil real es que no dispone de espectro. A costa de una mayor inversión, es la modalidad con mayor control sobre el negocio: puede terminar llamadas –con lo que aparecen nuevas oportunidades de beneficio por las diferencias entre tráfico entrante y saliente–, escoger de forma flexible el operador proveedor sin necesidad de enviar nuevas tarjetas SIM a los clientes en caso de cambio, y goza de casi tantas posibilidades de innovación como un operador completo.
- Un Operador de Servicio (*Service Operator*) es menos independiente que el OMV completo, encargándose de los puntos 4-6. Es decir, debe mantener todos los sistemas de TI, incluyendo las interfaces con el Operador Móvil Real, que pueden ser un reto especialmente cuando éste utiliza sistemas antiguos.
- Un *reseller*, que se ocupa únicamente del punto 6, es decir, de las relaciones con el cliente. Es el modelo que ofrece una entrada más directa al mercado, especialmente si se llega a un acuerdo con un MVNE (*Mobile Virtual Network Enabler*), que se encargaría de los puntos 2-5.

Dependiendo del modelo de negocio, y los plazos, inversiones y beneficios esperados, se escogerá un modelo u otro. [39]

5.2. Arquitectura de sistemas

República Móvil, como *reseller*, tiene dos principales interfaces de entrada y salida de datos de clientes. Los datos de todo cliente cuya alta finalice deben ser enviados al sistema del MVNE, que a su vez se intercambiará información con el Operador Móvil Real.

Además de los sistemas esenciales para que el cliente pueda disfrutar del servicio de comunicaciones móviles, hay otros sistemas empleados para facilitar las relaciones con el cliente, la gestión de incidencias y las funciones del *call center*.

5.2.1. Servidores

Los clientes, los agentes de *call center* y los distribuidores recopilan información y realizan los procesos de alta a través del portal web. Por ello, es necesario tener una infraestructura fiable

que permita minimizar los tiempos de indisponibilidad en caso de ser necesario realizar algún cambio.

Para cumplir este requisito, se han desplegado dos servidores web productivos, cuya carga se balancea, y un servidor de pre-producción. Los dos primeros, conectados a la base de datos `rm_walva`, son copias exactas, y son los visibles a través de `www.republicamovil.es`. El tercero de ellos, que emplea la base de datos `rm_walva-pre`, es el empleado a la hora de implementar nuevas funcionalidades y realizar pruebas antes de liberarlas a los clientes. Una vez que todo está correcto, se procede a realizar un paso a producción en un momento del día de mínima afluencia de visitantes.

Además, hay varios servidores web internos, de menor capacidad. En primer lugar, nos encontramos con el servidor de xCally, software VoIP basado en Asterisk diseñado para *call centers* que incluye reportes en directo vía un servidor HTTP integrado en él. En segundo lugar, hay un servidor de MySQL que aloja todas las bases de datos utilizadas. En tercer lugar, está el servidor de OTRS, que aloja el software empleado para la gestión de incidencias. En cuarto lugar, la máquina de Pentaho es un servidor Linux CentOS que contiene tanto una instancia de Pentaho BI Server como Pentaho Kettle y permite lanzar procesos ETL desde línea de comandos. Es sencillo automatizar estos procesos mediante *bash scripts* y el comando `cron`. Por último, se instaló un servidor WAMP (Windows + Apache + MySQL + PHP) sobre una de las máquinas del *call center* para alojar herramientas de uso interno: gestión de inventario, facturación, etc.

Por otro lado, se emplean ciertos recursos externos. Entre los más importantes destacan los sistemas del MVNE y LiveHelpNow, servicio de chat en tiempo real integrado en el portal web.

5.2.2. Bases de datos

Las bases de datos más importantes entre las empleadas son las bases de datos `rm_walva` y `rm_walva-pre`, de producción y pre-producción, y las bases de datos `rm_mvno`, `rm_mcdf` y `rm_xfac`, réplicas de la información del MVNE.

5.2.2.1. `rm_mvno`

La base de datos `rm_mvno` contiene una réplica de la información empleada por el MVNE, que a su vez se comunica con el Operador Móvil Real. Dicho de otra forma, es la base de datos que contiene la información productiva real de los clientes. Las tablas más importantes son las siguientes:

- `rm_clientes`: contiene información básica sobre los clientes: datos personales básicos, fechas de alta y baja, estado, y *scoring*, indicando éste último si el alta del cliente ha sido o no aceptado, de acuerdo con las políticas de la compañía.
- `rm_msisdn`: permite realizar un seguimiento básico sobre todas las líneas: alberga datos como el IMSI, ICCID, estado actual, estado de *roaming*, y las fechas de alta, último estado, última conexión y última llamada.
- `rm_clientes_estados` y `rm_msisdn_estados`: asocian a cada código de estado de línea y cliente una descripción de éste.
- `rm_productos`: mantiene todos los productos (tarifas) que comercializa el MVNO, junto al bono que llevan por defecto. Cada uno de estos bonos aparece en `rm_productos_bonos`, donde se indica qué tipo de consumo incluye (voz, datos y/o SMS) y a qué precio.

- **rm_alta_portabilidad**: guarda información de cada alta por portabilidad que ha recibido el MVNO, incluyendo datos básicos de la línea, el operador donante, y un histórico de los estados por los que ha pasado el MSISDN en el Nodo Central de Portabilidad.

El MVNE es el encargado de mantener la información de estas tablas actualizadas. La frecuencia de actualización se decide en función de cada tabla, siempre teniendo en cuenta que la carga de datos inhabilita momentáneamente la lectura de ésta.

Además de las tablas señaladas, existen otras como **rm_con_pasarela_activations**, **rm_con_pasarela_activations_callback** y **rm_con_pasarela_activations_interact** que almacenan las interacciones entre el servidor web de República Móvil y el *web service* de activación del MVNE.

5.2.2.2. **rm_mcdf**

La base de datos **rm_mcdf** contiene todos los CDR (*call detail records*) generados por los clientes del MVNO. El MVNE crea, cada día, una tabla de la forma **rm_cdrsyyyymmdd**, donde **yyyymmdd** contiene la fecha. Los datos más relevantes almacenados son los siguientes:

- El ID y MSISDN del cliente.
- El tipo de CDR: voz, datos o SMS.
- El número con el que se ha comunicado. Este campo estará vacío en caso de referirse a un consumo de datos.
- Fecha y hora del CDR.
- Duración de la llamada.
- Bytes de subida y bajada transmitidos.
- Precio según tarifa.
- Bonos aplicados.
- Precio tras aplicar bonos.

La tabla de cada día se sigue actualizando hasta cinco días más tarde, ya que los operadores extranjeros tardan un tiempo en enviar al proveedor los CDRs de los clientes en *roaming*. Por ley, todo operador telefónico está obligado a almacenar estos datos entre 6 y 24 meses. [40]

5.2.2.3. **rm_xfac**

La base de datos **rm_xfac** contiene los resultados del proceso de facturación llevado a cabo por el MVNE. Cada mes, se crea una tabla **cdrs_yyyyymm** con todos los CDRs registrados en ese período. Además, se actualizan las siguientes tablas:

- **historico_facturacion**: incluye datos para la generación de cada factura, así como su importe.
- **historico_voz**: desglosa el consumo de cada factura según su tipo.

- **historico_datos**: almacena para cada factura y producto el importe de la cuota y el del consumo total.
- **historico_descuentos**: incluye los descuentos, promociones y compensaciones asociados a cada factura.
- **Recibos**: relaciona cada factura con su estado de pago.

5.2.2.4. `rm_walva`

La base de datos `rm_walva` es la que almacena todos los datos que maneja el MVNO por sí mismo. En primer lugar, contiene un almacén de datos normalizado que contiene todas las órdenes tramitadas a través del portal web. Entre las tablas más importantes, relacionadas entre sí a través de la tabla `FullOrder`, destacan:

- **CustomerOrder**: almacena el estado de cada subproceso de la orden: alta, pago y envío.
- **FullClientData**: mantiene los datos personales del cliente tal y como éste los ha introducido en la web.
- **AddressData**: es donde se conservan los datos de la dirección del cliente. Una misma orden puede generar dos filas en aquellos casos en que el cliente tenga una dirección de facturación distinta a la de envío.
- **NumberPortDetailsData**: tabla donde se insertará una fila únicamente cuando el cliente esté portando desde otra compañía.
- **LogisticOrder**: almacena detalles sobre el envío, como el estado y el número de seguimiento.
- **Rate**, que incluye las tarifas, junto a sus descripciones y precios, que serán mostrados en el portal web.
- **Channel** y **SFID**: contienen información sobre los distribuidores.

Además de los datos introducidos por los clientes, República Móvil cuenta con una serie de procedimientos almacenados con los que se calculan ciertos datos a medida. Diariamente, se comprueba qué clientes cumplen las condiciones del *Plan Pioneros*, actualizando las tablas `mgm_clientes`, `mgm_entidades`, `mgm_datos_mes` y `mgm_relaciones`. Cuando finaliza el período de facturación, se calculan los descuentos a recibir por parte de cada cliente, actualizándose las tablas `mgm_saldo_pioneros`, `mgm_importe_acum` y `mgm_historico_datos_mes`.

Por otro lado, los datos de facturación creados por el MVNE se procesan de acuerdo con los requerimientos definidos por el Departamento de Marketing antes de ser mostrados al cliente. El código de la web recoge automáticamente estos datos una vez se cargan en las tablas:

- `fac_cli_datos_unicos`, que mantiene datos a nivel de cliente;
- `factura_cli_datos_msisdn`, con datos a nivel de MSISDN;
- `fac_consumo_resumido`, con detalles sobre cada tipo de consumo realizado por el MSISDN;

junto a otras tablas menos relevantes.

5.2.3. CRM y gestión de incidencias

El encargado de mantener el software CRM (*Customer Relationship Management*) empleado en República Móvil, denominado Xena, es el MVNE. Este programa es un panel de control que permite administrar todos los datos de clientes y líneas, asignar un *score* de riesgo y aceptar o denegar altas en función de éste, controlar su consumo, modificar el estado de cada línea, y elevar incidencias al MVNE. También es posible realizar altas desde la aplicación, pero se desestimó el uso de esta función en favor del portal web, con el objetivo de uniformizar el flujo de alta.

Xena utiliza directamente la base de datos del MVNE. Los cambios efectuados desde la aplicación tienen, por tanto, efecto instantáneo sobre las líneas de los clientes. Sin embargo, no se reflejan en la base de datos `rm_mvno` hasta que se realiza la copia.

Por otro lado, para gestionar las incidencias y comunicaciones con los clientes a nivel local, se utiliza OTRS, un software *Help Desk* de código abierto. Esta aplicación genera un ticket de forma automática cada vez que un cliente se pone en contacto con el operador a través de cualquiera de los medios disponibles, asignándosele una categoría según el medio por el que contacte.

Los operarios del *call center* pueden también crear y elevar tickets a otro departamento en caso de requerirlo, y emplear OTRS para comunicarse con los clientes por correo electrónico.

5.3. Procesos

5.3.1. Alta de cliente

El proceso de alta está diseñado para llevarse a cabo a través del portal web, incluso cuando el cliente contrata por vía telefónica o se persona en un distribuidor. El proceso completo está representado en la figura 5.2.

La página principal del portal da a elegir una de las tarifas a contratar. En cuanto el cliente escoge una, aparece un formulario en el que introduce su nombre, dirección de correo electrónico y teléfono de contacto. Estos datos se almacenan en las tablas `PotentialClientData` y `PotentialOrder`, versiones con menos campos de `FullClientData` y `FullOrder`, ya que en este punto se tiene menos información.

A continuación, se solicita al cliente que introduzca datos de pago, que se registran junsto a las direcciones de envío y facturación en `FullClientData`, `AddressData` y `NumberPortDetailsData`. Dado que sus identificadores son claves foráneas en `FullOrder`, las inserciones deben ser realizadas en estas tablas antes de ser relacionadas en `FullOrder`. En este punto, el cliente deja de ser un cliente potencial, por lo que la tabla `PotentialOrder` se actualiza de forma que almacene el nuevo identificador asignado al cliente en `FullClientData`.

El cliente puede escoger el medio a través del cuál abonar el importe de los gastos de gestión y envío. Si escoge realizar el pago mediante tarjeta de crédito será redirigido a un servicio externo denominado TPV (Terminal Punto de Venta) Virtual. De no ser así, sus datos quedarán registrados para realizar un cargo a cuenta una vez haya sido enviada la tarjeta SIM.

Una vez se han guardado correctamente todos los datos se procede a enviar una petición de alta al *web service* del MVNE mediante un fichero XML que contiene los datos del cliente. El *web service* validará los datos contra la base de datos del MVNE, pudiendo aceptar o denegar el alta por diversos motivos.

Un caso especial en este punto del proceso es aquel en el que un cliente decide realizar una portabilidad desde otro MVNO al cuál da servicio el mismo MVNE que a República Móvil. En

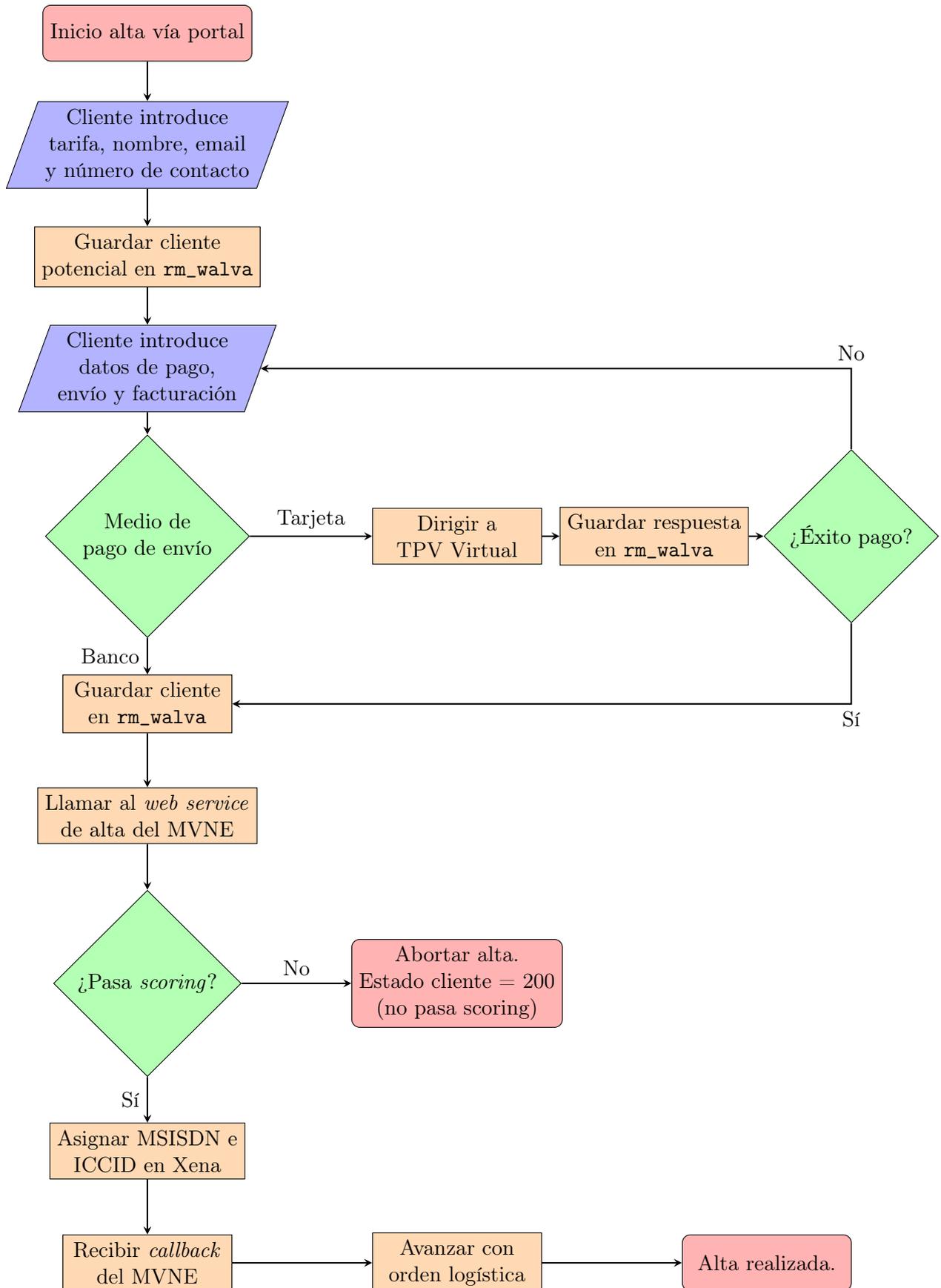


Figura 5.2: Proceso de alta de un cliente

ese caso el cambio de compañía es transparente de cara al Nodo Central de Portabilidad, ya que la visibilidad de éste se limita al MVNE, que se encargará internamente de mover al cliente de compañía.

En el momento en el que el *web service* recibe la nueva línea, ésta aparece en Xena con estado "Pendiente de *scoring*". Un operador de *back office* se encargará de validar la solicitud y aprobarla en función de los criterios operativos definidos por la compañía. En caso de superar esta fase, el *web service* del MVNE responderá al servidor web con el Id de cliente y MSISDN asignados, encargándose éste de actualizar el registro correspondiente en `FullClientData` y de hacer progresar la orden logística. En ese momento, se generarán todos los documentos necesarios, y el pedido estará preparado para ser empaquetado y enviado.

El MVNE, por su parte, se encargará de activar la línea, si se trata de un alta nueva, o de realizar las peticiones correspondientes al Nodo Central de Portabilidad, y gestionar la activación en cuanto el proceso de portabilidad se haya completado.

5.3.2. Facturación

Si bien el encargado del proceso de facturación es el MVNE, el MVNO tiene que ocuparse de algunos aspectos de éste. El proceso de facturación se lleva a cabo mediante procedimientos almacenados SQL desarrollados internamente, empleando la base de datos de pre-producción.

Los sistemas del MVNE admiten descuentos y promociones, pero no están preparados para realizar los cálculos correspondientes al Plan Pioneros. Diariamente, se calcula el descuento de cada cliente en función de las líneas que lo han marcado como padrino. Cuando llega el final del mes, se procede a *congelar* el plan pioneros: se realiza una copia de las tablas de pioneros a la base de datos `rm_walva-pre`, y se procede a lanzar los procedimientos diarios por última vez, actualizando el número de ahijados de cada línea.

En cuanto el MVNE termina el proceso de facturación y carga los resultados en `rm_xfac` se puede proceder con el proceso interno, que limpia y agrega los datos de consumo conforme a las especificaciones del Departamento de Marketing, buscando que la factura sea lo más clara posible para el cliente. La carga se realiza en las tablas de facturación descritas en la sección 5.2.2.4.

Después de realizar un primer cuadro por parte del Departamento de Sistemas, se procede a finalizar el proceso del Plan Pioneros. El MVNE hace que la factura tenga valor 0.01 Euros cuando los ingresos del plan superan los gastos del cliente, por lo que es necesario lanzar un procedimiento que encuentre la diferencia y la transforme en saldo de la tarjeta monedero. Estos resultados se guardan en `mgm_saldo_pioneros`.

Finalmente, las facturas resultantes son validadas por los departamentos de Operaciones y Finanzas, liberándose a los clientes en cuanto se cuenta con su aprobación.

5.3.3. Cambios de titular

Los cambios de titular deben realizarse a través de Xena para que tengan efecto sobre los datos reales que mantiene el MVNE. Sin embargo, los datos no se actualizan en `rm_walva`, por lo que debe lanzarse un procedimiento que los corrija.

Buscando un compromiso entre mantener la trazabilidad del ID de cliente anterior y tener el menor impacto sobre las consultas SQL subyacentes del portal, la opción escogida es la de añadir un carácter al final del campo `Id_cliente` antiguo. De este modo, una simple cláusula `inner join` contra `rm_clientes` hará que el identificador antiguo deje de aparecer en los resultados productivos.

6

Estado del arte en predicción de impagos en telefonía

El fraude en telecomunicaciones se define como el robo de servicios de telecomunicación o el uso de un servicio de telecomunicación para cometer otros tipos de fraude. [41]

Weiss [42] define el fraude de suscripción como aquel en el que un cliente se da de alta en un servicio sin intención de pagarlo, mientras que Shawe-Taylor et al. [43] interpretan este término como el tipo de fraude en el que incurre una persona que utiliza una identificación falsa para acceder a un servicio, ya sea para su uso personal o para obtener líneas con las que ofrecer llamadas baratas de larga distancia, con el consiguiente beneficio económico. Además del fraude de suscripción, nombran otros tipos relevantes como el basado en llamadas *premium*, en el retardo que se produce entre la creación y el procesado de un CDR cuando la línea se encuentra en *roaming*, o el robo de terminales. Todos ellos tienen en común el impago de los servicios.

En 1996, Ezawa y Norton [44], de AT&T, identificaron el gran tamaño del *dataset*, los distintos costes de la clasificación incorrecta, el alto número de variables y la naturaleza probabilística del comportamiento como los grandes retos del problema de identificación de clientes que evaden el pago. Por ello, escogieron emplear redes bayesianas sobre un *dataset* compuesto por llamadas acompañadas de un resumen de datos del cliente que las realiza, siendo un 10% de ellas impagadas. Definiendo las llamadas impagadas como positivas, llegan a conseguir un 36.57% de verdaderos positivos y un 9.11% de falsos positivos.

Daskalaki et al. [45] diseñaron en 2003 un sistema cuyos principales objetivos eran detectar tantos clientes insolventes como fuera posible, minimizar las falsas alarmas y avisar a tiempo al proveedor de servicio de forma que tome las acciones necesarias ante un impagador. Emplearon un *data warehouse* que albergaba datos de 100.000 clientes a lo largo de 17 meses, donde un 0.7% era insolvente en cada período de facturación, si bien remuestrearon el *dataset* para obtener un 10% de impagos. Partieron de 47 variables de datos estáticos y CDRs de clientes, que fueron reducidas a 16 mediante análisis discriminante, y utilizaron redes neuronales y árboles de decisión sobre ellas. Encontraron que el segundo método daba mejores resultados, con un 59.38% de sensibilidad frente al 37.50% obtenido con redes neuronales.

Pinheiro et al. [46] trataron en 2006 de encontrar un modo de evitar pérdidas en los ingresos de un operador fijo brasileño causadas por aquellos clientes que, aun no teniendo intención de incurrir en fraude, retrasaban el pago de sus facturas. Partiendo de una base de datos de 5 millones de cliente realizaron, en primer lugar, un estudio de los distintos tipos de insolvencia empleando mapas autoorganizados de Kohonen como algoritmo de *clustering*, encontrando cinco

grupos de clientes en función de su consumo y comportamiento de pago. A partir de estos resultados, definieron los clientes de cada *cluster* como *buenos* (72% de los clientes) y *malos* (28% de los clientes), dando la posibilidad a la compañía de darles distinto tratamiento a la hora de tratar de recaudar el dinero que se le debe. En segundo lugar, ya etiquetados los clientes en función de su compromiso ante el pago, implementaron un clasificador utilizando un *bagging* (agregación de *bootstrap*) de clasificadores basados en redes neuronales, dividiendo previamente el *dataset* en 10 partes empleando mapas autoorganizados de Kohonen con la intención de que cada clasificador trabajara con un *dataset* más homogéneo. Así, consiguieron precisiones del 83.95% y 81.25% para las clases *bueno* y *malo*, respectivamente.

En 2012, Ząbkowski et al. [47] utilizaron datos de 94.620 usuarios (donde un 12% era insolvente) de un operador móvil polaco para desarrollar un modelo con objetivos similares a los de Daskalaki et al., si bien emplearon 205 variables estáticas y de comportamiento de los clientes, que fueron reducidas a 26 realizando pruebas *t* de Student entre ellas. Obtuvieron una AUC de 88.29% y 85.16% mediante modelos de redes neuronales y árboles de decisión, respectivamente, si bien achacan el peor rendimiento de éstos últimos a los cambios temporales del mercado. Además, argumentan que una de las limitaciones de su modelo es que no tiene en cuenta los motivos de la insolvencia, ya que esta puede ser o no intencionada.

Autor	Técnica	Sensib.	Especif.	g-means	AUC
Ezawa et al.	Redes bayesianas (umbral 0.5)	36.57%	90.89%	57.65%	–
Ezawa et al.	Redes bayesianas (umbral 0.7)	25.29%	95.55%	49.16%	–
Daskalaki et al.	Redes neuronales	37.50%	98.32%	60.72%	–
Daskalaki et al.	Árboles de decisión	59.38%	98.78%	76.58%	–
Pinheiro et al.	Bagging de redes neuronales	92.38%	84.45%	88.33%	–
Ząbkowski et al.	Redes neuronales	–	–	–	88.29%
Ząbkowski et al.	Árboles de decisión	–	–	–	85.16%

Tabla 6.1: Resumen de estudios sobre insolvencia en telecomunicaciones móviles

Además de los artículos recogidos en esta sección, que analizan problemas muy similares al de este proyecto –si bien todos tienen en cuenta datos de consumo de los clientes, mientras que este proyecto estudia el punto en el que en general no se tiene conocimientos previos sobre los clientes–, hay numerosos estudios sobre riesgo de crédito en el ámbito bancario, donde se evalúa la solvencia de un prestatario; así como de fraude en transacciones de tarjetas de crédito. Aunque los problemas son bastante semejantes en apariencia, hay diferencias importantes en el impacto económico del impago, además de que las entidades implicadas cuentan con mucha más información relevante que un operador móvil.

También existen artículos sobre distintos tipos de fraude telefónico, como la comparativa de Sallehuddin et al. [48] entre redes neuronales y SVM para el problema de detección de *SIM box fraud*, compras masivas de tarjetas SIM para ofrecer llamadas a bajo coste, donde se ve que SVM mejora el rendimiento de redes neuronales.

7

Diseño del *data warehouse*

El primer paso a la hora de implementar un *data warehouse* es realizar un diseño tentativo de las tablas que va a contener. Debido a su versatilidad a la hora de implementar nuevos *datamarts*, la arquitectura escogida es la mostrada en la figura 3.1.

La metodología de desarrollo empleada está basada en la de Bill Inmon, recogida en la sección 3.3. El NDS contiene toda la información que puede ser requerida por los futuros *datamarts*, permitiendo únicamente cierta redundancia en los datos en aquellos casos en los que su mantenimiento tenga sentido funcional o simplifique el diseño en suficiente medida.

En contraposición al modelo generalizado que sigue el NDS, el DDS diseñado cumple todos los requisitos funcionales definidos por los departamentos que lo emplearán para controlar la solvencia de los clientes. Siguiendo el método Kimball, se busca la facilidad de uso y claridad de los datos incluidos en este almacén, por lo que se permite cierta redundancia.

El primer proceso ETL carga los datos de los sistemas operacionales en el NDS manteniéndose lo más simple posible, realizando comprobaciones básicas de la integridad de los datos, y sin apenas transformarlos. El segundo ETL es el que realiza las modificaciones oportunas en los datos de acuerdo con las peticiones de los usuarios. Al tratarse de un proceso más específico, puede ser más complejo, balanceándose así el riesgo del proyecto entre los dos ETL.

Dado que se ha empleado MySQL, todas las tablas del *data warehouse* son MyISAM y siguen las recomendaciones y convenciones expresadas en la sección 2.3, salvo cuando se especifique lo contrario.

7.1. NDS (*Normalized Data Store*)

El almacén normalizado de datos mantiene toda la información de los diversos sistemas operacionales de la compañía de potencial utilidad en el sistema de BI. De acuerdo con los requerimientos, estos datos provienen de:

- los datos mostrados en el portal web, que se mantienen en `rm_walva`.
- los datos productivos de la réplica del MVNE, incluyendo:
 - los datos oficiales de los clientes, almacenados en `rm_mvno`.

- los registros de llamadas, disponibles en `rm_mcdf`.
- las facturas calculadas por el MVNE, que pueden encontrarse en `rm_xfac`.

Estas bases de datos operacionales se describieron en detalle en la sección 5.2.2.

Analizando los datos a almacenar, se encuentra que pueden dividirse en cinco grandes bloques, que establecen el prefijo de las tablas del NDS:

- `cliente_`: contienen información sobre los clientes, sus líneas, y cómo participan en el Plan Pioneros.
- `prov_`: compuestas por datos de los proveedores que proporcionan de alguna forma clientes a la compañía, ya sean distribuidores o empresas que hayan participado en alguna acción comercial.
- `factura_`: almacenan el detalle de cada factura, incluyendo importes de cuota, consumos y promociones, así como algunos datos del cliente asociados a la propia factura.
- `cdr_`: relacionadas con los CDR.
- `info_`: tablas adicionales que contienen información estática relacionada con las otras tablas.

A continuación se analizan en detalle las tablas de cada grupo.

7.1.1. `cliente_`

La figura 7.1 recoge todas las tablas con prefijo `cliente_` incluidas en el NDS. La tabla principal es `cliente_datos`, que contiene todos los datos personales del usuario, incluyendo sus direcciones de envío y facturación, el número de acciones que ha sufrido a través del CRM y su estado.

Cada cliente puede tener múltiples líneas a su nombre, registrándose éstas como filas en la tabla `cliente_linea`, que contiene información estática y dinámica del MSISDN: datos técnicos como el IMSI y el ICCID, el producto contratado, información sobre el proceso de alta, el padrino solicitado en el Plan Pioneros y el estado de esta solicitud, y datos básicos de uso, así como su estado.

Los datos del plan pioneros se calculan mes a mes a nivel de cliente, como se explicó en la sección 5.3.2. Estos datos mensuales se almacenan en `cliente_pioneros`. De alcanzar el usuario el saldo mínimo para recibir una tarjeta monedero, aparecerán los datos de ésta en `cliente_pioneros_monedero`.

Todas las tablas están diseñadas para mantener un histórico de las bases de datos operacionales, excepto las que mantienen las descripciones de los estados, que son estáticos. Por tanto, se han incluido los campos `version`, `date_from` y `date_to`, que son empleados por el ETL para tener constancia de cuál es el dato válido en cada fecha. Por este motivo, las relaciones 1:1 y 1:N mostradas en la imagen son conceptuales, ya que se asume que únicamente se está tomando la última versión del registro.



Figura 7.1: Esquema de las tablas de tipo cliente_

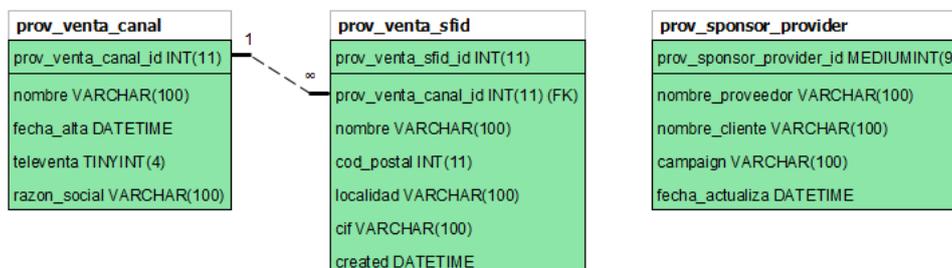


Figura 7.2: Esquema de las tablas de tipo prov_

7.1.2. prov_

Las tablas de tipo `prov_`, mostradas en la figura 7.2, mantienen información sobre los distribuidores y proveedores.

Por un lado, `prov_venta_canal` contiene los datos de los distintos canales de venta, además de un *flag* que indica si este canal es de televenta o se trata de un distribuidor físico. Cada una de las tiendas de ese distribuidor irá incluida en la tabla `prov_venta_sfid`.

Por otro lado, la tabla `prov_sponsor_provider` almacenará los distintos proveedores y campañas de captación acordadas con ellos, indicando el cliente su pertenencia a estas promociones incluyendo determinado código cuando se le solicita el número de teléfono de su padrino en el momento del alta.

7.1.3. factura_

La figura 7.3 muestra las tablas de tipo `factura_`. En este caso, la tabla principal es `factura_cliente`, que contiene todos los datos del cliente incluidos en la factura. Su conservación es necesaria para mantener la trazabilidad, y poder regenerar una factura de ser necesario.

La información básica de cada una de las líneas facturadas se almacena en `factura_linea`, mientras que un resumen de los CDRs se mantiene en `factura_linea_consumos`.

Los descuentos realizados en cada período de facturación permanecen en las tablas `factura_cliente_pioneros`, `factura_promocion` y `factura_promocion_promo`.

7.1.4. cdr_

Las tablas `cdr_` incluyen todos los *call detail registers* de los clientes de la compañía. Para poder manejar el gran volumen de la tabla `cdr_historico`, se ha particionado por fecha, siendo necesario añadir este campo a la clave primaria. Cada partición contiene los CDRs de un mes.

Código 7.1: Fragmento de la definición del particionado de la tabla `cdr_historico`

```
1 ROW_FORMAT = FIXED PARTITION BY RANGE(TO_DAYS(Fecha)) PARTITIONS 38(  
2 PARTITION part0 VALUES LESS THAN (735538),  
3 PARTITION part1 VALUES LESS THAN (735568),  
4 PARTITION part2 VALUES LESS THAN (735599),  
5 /*...*/  
6 PARTITION part36 VALUES LESS THAN (736634),  
7 PARTITION part37 VALUES LESS THAN (736664))
```

7.1.5. info_

Por último, como se muestra en la figura 7.5, las tablas `info_` mantienen datos adicionales que pueden ser empleados por cualquiera de las otras tablas.

7.2. DDS (*Dimensional Data Store*)

El almacén de datos dimensional se ha diseñado siguiendo la metodología de Ralph Kimball, revisada en la sección 3.2.2.2. Se han identificado tres actores principales en el negocio y se ha

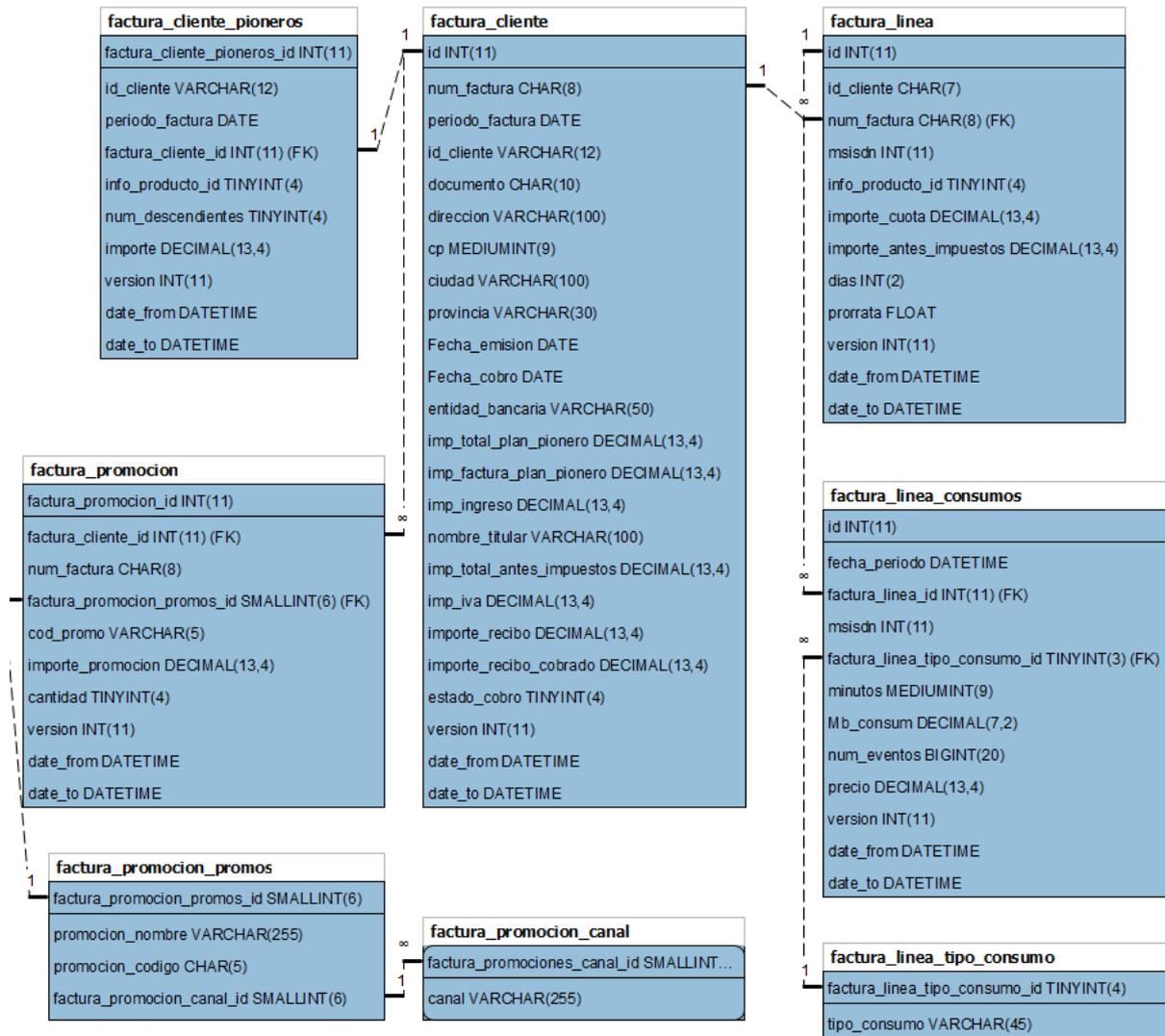


Figura 7.3: Esquema de las tablas de tipo factura_

definido una estrella para cada uno de ellos: líneas, facturas y CDRs. A continuación, como se muestra en el cuadro 7.1, se ha elaborado una matriz para encontrar las dimensiones comunes entre los procesos de forma que se puedan emplear dimensiones conformadas.

A continuación se revisarán en detalle las medidas y dimensiones de cada uno de los procesos.

7.2.1. Línea

El proceso *Línea* viene descrito en el *information package* que se recoge en las tablas 7.2 y 7.3. La información que se necesita obtener de las líneas es el número de MSISDN que hay dados de alta y los ingresos que provienen de éstos, pudiéndose agregar por las dimensiones que se explican a continuación.

La dimensión Fecha se genera de forma estática, rellenando la tabla con una fila para cada día en un período indicado. Contiene diversos formatos de fecha, facilitando la generación de informes de acuerdo con las preferencias de cada usuario. Asimismo, ofrece indicadores útiles, como si el día es festivo o fin de semana.

	Fecha	Cliente	Producto	Pioneros	Canal	Origen línea	Tipo SIM	Tipo pago	Estado línea	Tipo consumo	Número de MSISDN anteriores	Días de MSISDN anteriores	Estado de cobro
Línea	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Factura	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓
CDR	✓	✓	✓							✓			

Tabla 7.1: Matriz de dimensiones conformadas

La dimensión Pioneros ofrece información sobre el grado de participación del cliente en dicho plan, ya sea recibiendo descuentos porque otros usuarios lo han marcado como padrino o escogiendo a otro cliente como padrino.

Por otro lado, la dimensión Cliente contiene datos personales básicos del titular de la línea. Además, contiene medidas de similitud entre las cadenas de texto que contienen el titular de la línea, de la cuenta bancaria y del receptor del envío. Se han mantenido tanto el valor numérico como un valor discretizado, de forma que se puedan satisfacer desde esta etapa los requerimientos del usuario y de la aplicación de *data mining*.

La dimensión Producto, por su parte, almacena la información de las tarifas disponibles en el operador: los minutos de voz y megabytes de datos incluidos, los que se ofrecen una vez superado el límite, y el *flag Throttle* que indica en qué casos se cobran los datos adicionales, o se permite el uso gratuito de datos extra a menor velocidad.

Las dimensiones Canal y Origen línea dan información sobre la procedencia del MSISDN, y el método a través del cuál el cliente ha realizado el proceso de alta.

Además de las citadas, hay cinco dimensiones que pueden ser expresadas mediante una única columna. Tres de ellas se han agrupado en una Dimensión Miscelánea, que también será empleada en otra estrella. Las dos restantes, correspondientes al historial previo del cliente en la compañía, se han almacenado por simplicidad en la tabla de hechos.

La implementación en base de datos se muestra en la figura 7.6.

7.2.2. Factura

En cuanto a las facturas, se quiere obtener la información detallada de importes y descuentos con respecto al tipo de consumo y al estado del cobro del recibo, además de las dimensiones ya vistas en el IP Línea. Los *information package* de Factura se describen de forma detallada en los cuadros 7.4 y 7.5.

La tabla de hechos `fact_factura` contiene una línea por número de factura, MSISDN, tipo de consumo y período.

La implementación en base de datos se muestra en la figura 7.7.

Tabla 7.2: *Information package* de Línea (parte 1)

Fecha	Cliente	Producto	Pioneros
Fecha completa	ID cliente	Código	Número de hijos
Año	Fecha de alta*	Nombre	Número de nietos
Mes	Estado	Precio	<i>Flag</i> padrino
Día	Tipo documento	Minutos	Motivo denegación padrino
Día de la semana	Descrip. Documento	Megas	Tarjeta monedero
Día del año	Género	Megas extra	
YYYYMM	Rango edad	<i>Throttle</i>	
MMYYYY	Cód. nacionalidad		
<i>Flag</i> fin de semana	Continente		
<i>Flag</i> festivo	Cód. continente		
Descripción festivo	Provincia		
	Localidad		
	Código Postal		
	Proveedor email		
	Dominio email		
	Permite envío de información		
	Similitud titular banco y línea		
	Rango de sim. titular banco y línea		
	Similitud titular envío y línea		
	Rango de sim. titular envío y línea		

7.2.3. CDR

Por último, se requiere poder consultar diversos datos estadísticos sobre la duración y el importe de los CDR, siendo posible analizarlos en función del MSISDN, tipo de consumo, el cliente y el producto. El *information package* se muestra en el cuadro 7.6.

La tabla de hechos es una versión resumida de `historico_cdr`. Al no ser necesario tal nivel de detalle, los eventos se han agrupado por MSISDN, fecha y tipo de consumo.

La implementación en base de datos se muestra en la figura 7.8.

Tabla 7.3: *Information package* de línea (parte 2)

Canal	Origen línea	Dim. misc.	En tabla de hechos
Nombre tienda	Tipo de alta	Tipo de SIM	Días MSISDN anteriores
Código postal	Tipo de contrato anterior	Tipo de pago	Número de MSISDN anteriores
Localidad	Proveedor de origen	Estado de línea	
Provincia	Operador anterior		
Nombre canal			
<i>Flag</i> televenta			
Razón social canal			
Fecha de alta del canal*			

Hechos a medir: número de MSISDN (al agregar), importe facturado total

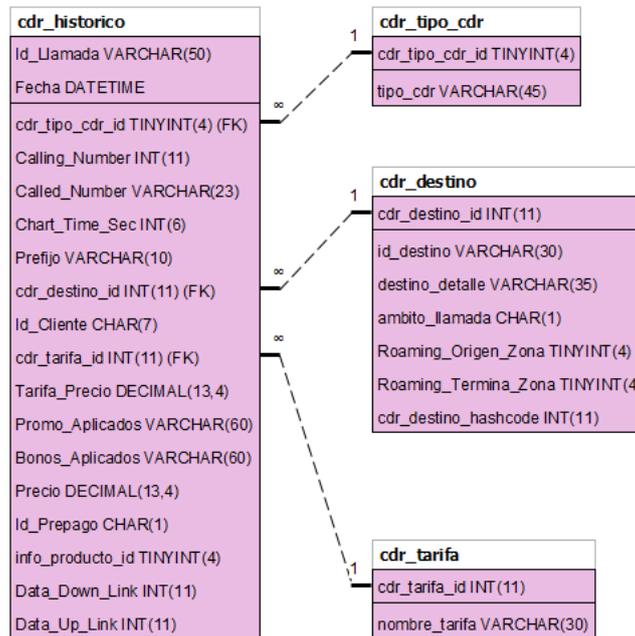


Figura 7.4: Esquema de las tablas de tipo cdr_

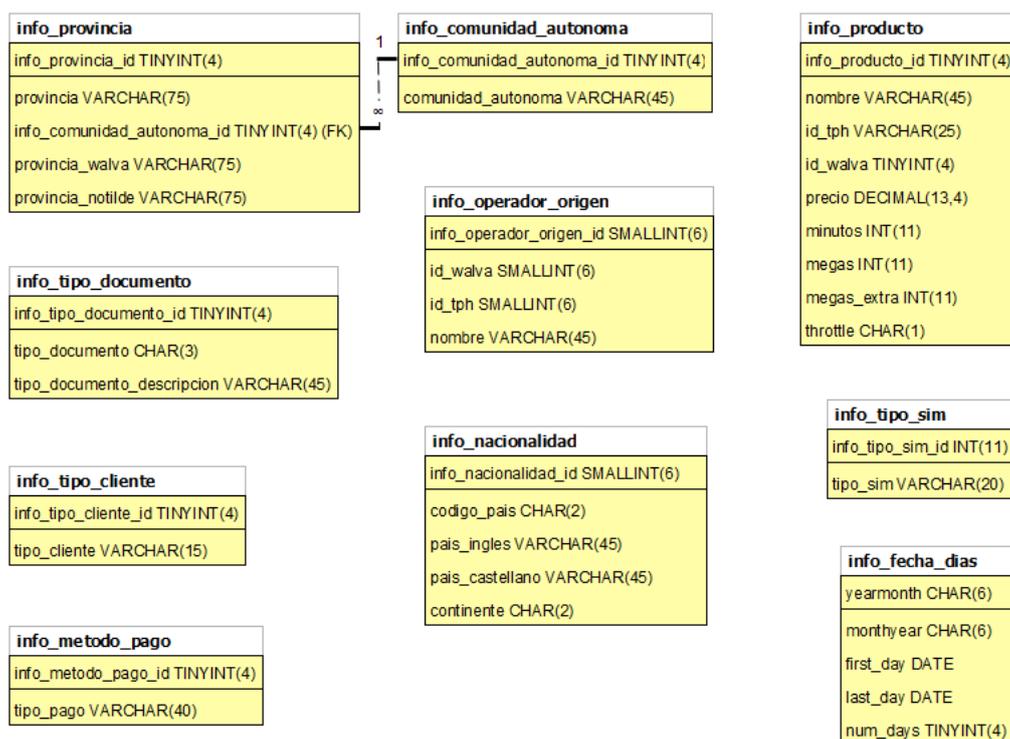


Figura 7.5: Esquema de las tablas de tipo `info_`

Tabla 7.4: *Information package* de Factura (parte 1)

Fecha	Cliente	Producto	Pioneros
Fecha completa	ID cliente	Código	Número de hijos
Año	Fecha de alta*	Nombre	Número de nietos
Mes	Estado	Precio	<i>Flag</i> padrino
Día	Tipo documento	Minutos	Motivo denegación padrino
Día de la semana	Descrip. Documento	Megas	Tarjeta monedero
Día del año	Género	Megas extra	
YYYYMM	Rango edad	<i>Throttle</i>	
MMYYYY	Cód. nacionalidad		
<i>Flag</i> fin de semana	Continente		
<i>Flag</i> festivo	Cód. continente		
Descripción festivo	Provincia		
	Localidad		
	Código Postal		
	Proveedor email		
	Dominio email		
	Permite envío de información		
	Similitud titular banco y línea		
	Rango de sim. titular banco y línea		
	Similitud titular envío y línea		
	Rango de sim. titular envío y línea		

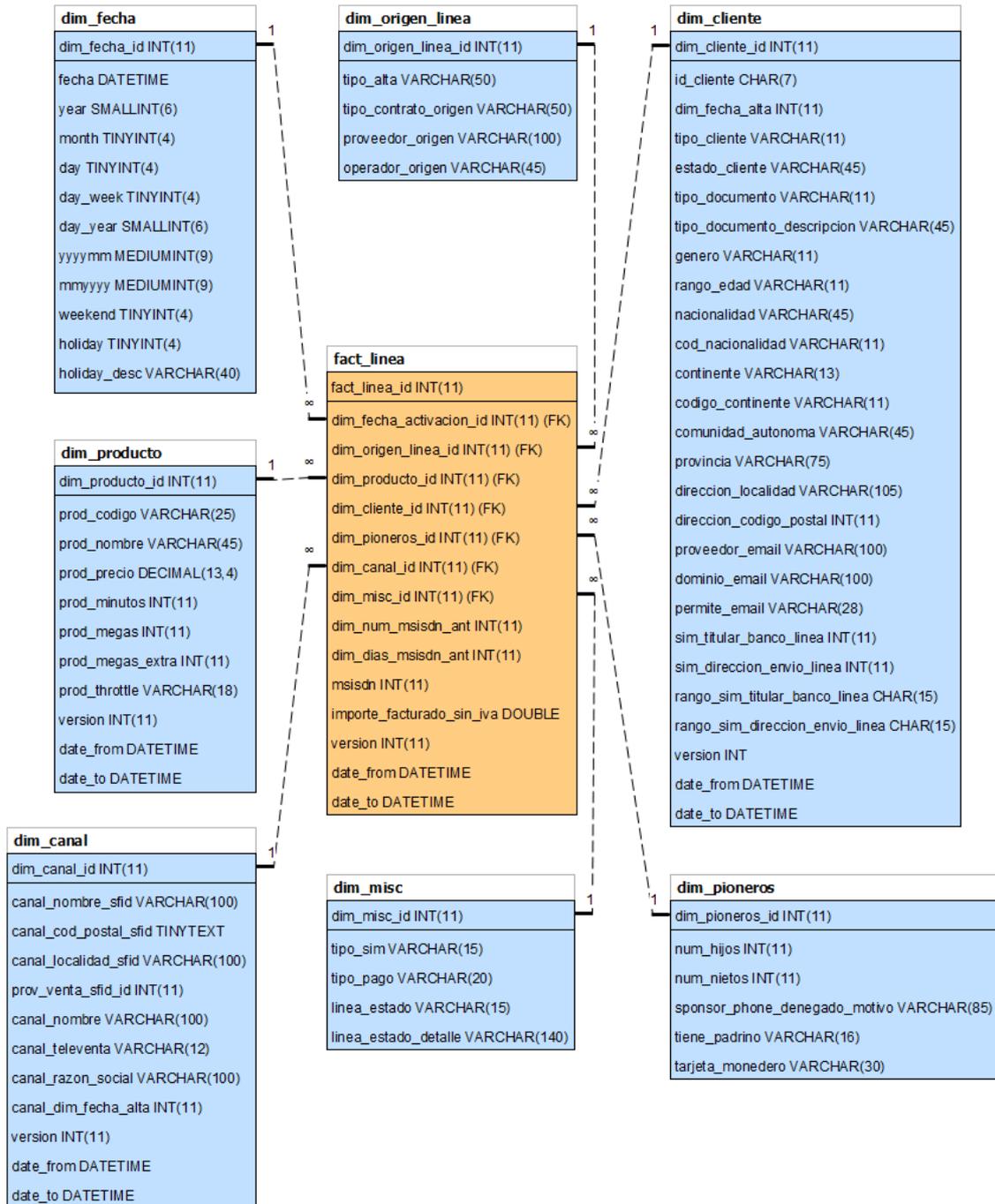


Figura 7.6: El *star schema* Línea

Tabla 7.5: *Information package* de factura (parte 2)

Canal	Origen línea	Tipo de consumo	Estado de pago
Nombre tienda	Tipo de alta	Estado de pago	Estado de pago
Código postal	Tipo de contrato anterior		
Localidad	Proveedor de origen		
Provincia	Operador anterior		
Nombre canal			
<i>Flag</i> televenta			
Razón social canal			
Fecha de alta del canal*			

Hechos a medir: Minutos consumidos, datos consumidos, importe consumos, número de eventos de consumo, importe antes de impuestos, descuento por plan pioneros, ingresos a saldo de plan pioneros, balance plan pioneros, importe a cobrar, importe cobrado, índice de consumo respecto a período anterior

Tabla 7.6: *Information package* de CDR

Fecha	Cliente	Producto	CDR Tipo consumo
Fecha completa	ID cliente	Código	Familia
Año	Fecha de alta*	Nombre	Grupo
Mes	Estado	Precio	Roaming
Día	Tipo documento	Minutos	Bonos
Día de la semana	Descrip. Documento	Megas	
Día del año	Género	Megas extra	
YYYYMM	Rango edad	<i>Throttle</i>	
MMYYYY	Cód. nacionalidad		
<i>Flag</i> fin de semana	Continente		
<i>Flag</i> festivo	Cód. continente		
Descripción festivo	Provincia		
	Localidad		
	Código Postal		
	Proveedor email		
	Dominio email		
	Permite envío info.		
	Similitud titular banco y línea		
	Rango de sim. titular banco y línea		
	Similitud titular envío y línea		
	Rango de sim. titular envío y línea		

Hechos a medir: Número de eventos, número de destinos, suma de importe, importe máximo de CDR, importe medio CDR, total de datos consumidos, total de minutos consumidos, cantidad máxima de datos en un CDR, duración máxima de CDR, media CDR datos, media duración CDR.

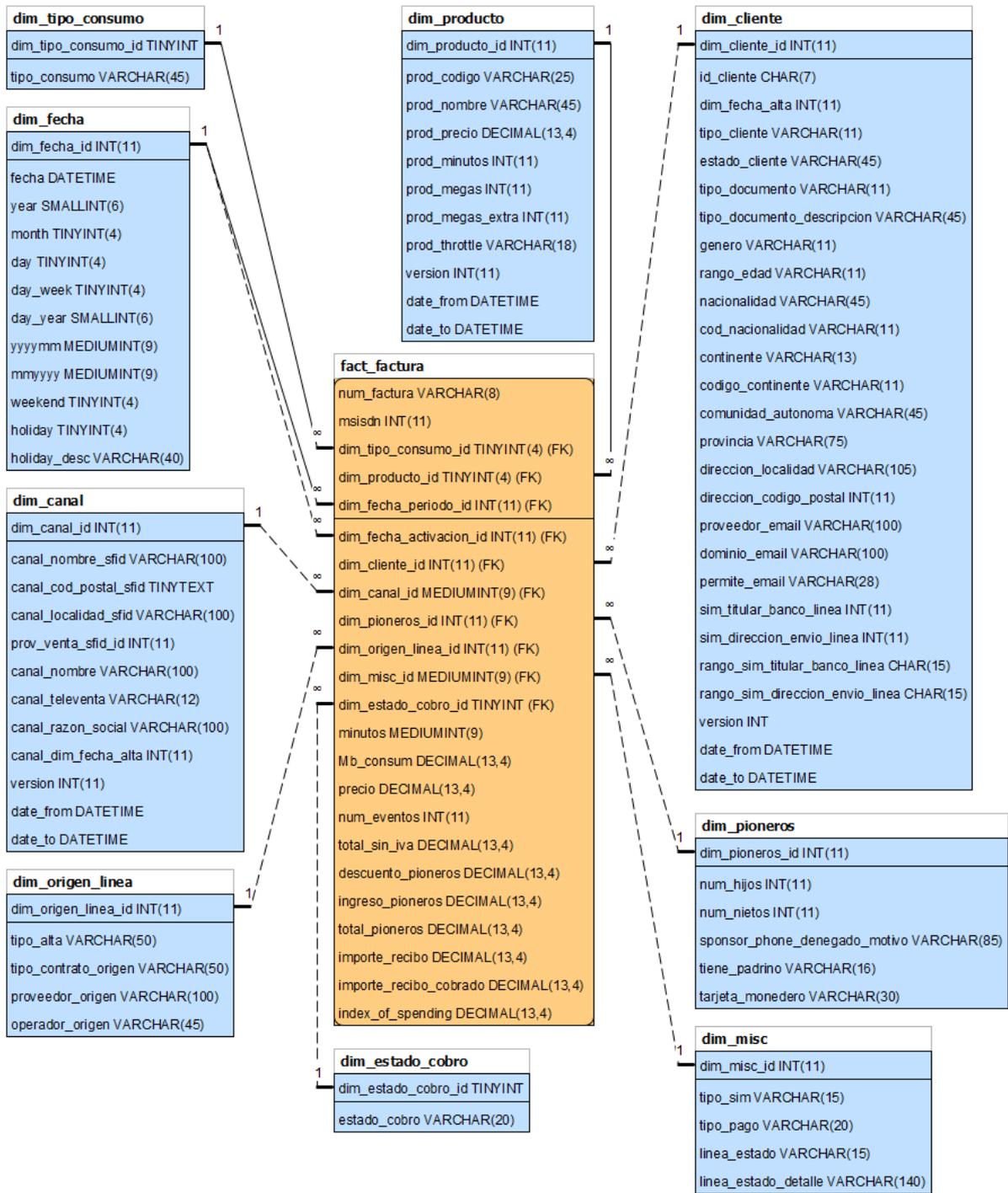


Figura 7.7: El star schema Factura

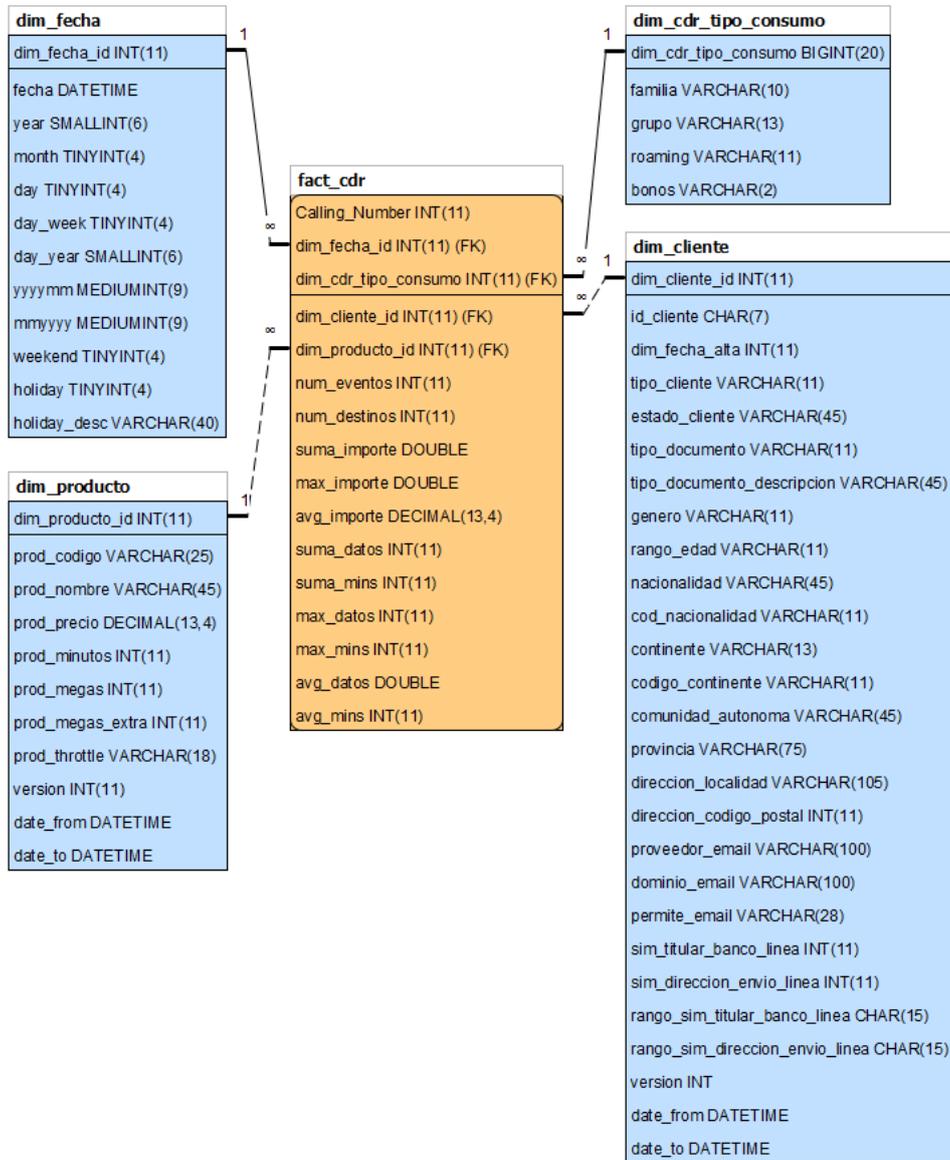


Figura 7.8: El *star schema* CDR

8

Diseño del ETL

Una vez se han creado las tablas del *data warehouse* en el servidor MySQL, deben cargarse los datos a través del software ETL. En este caso, se ha escogido la suite de código abierto Pentaho, que ofrece una solución ETL denominada Pentaho Data Integration, además de un servidor web de BI y Weka, una aplicación de *data mining*.

8.1. Conceptos básicos de Pentaho Data Integration

Pentaho Data Integration (PDI, o Kettle) es una solución ETL basada en metadatos. Permite programar procesos de extracción, transformación y carga de forma gráfica, ofreciendo más de 100 objetos en el momento de su instalación, incluyendo entradas, salidas y transformaciones. Además, puede ser ampliado mediante *plug-ins* disponibles en el Pentaho Marketplace.

Spoon, la GUI de Pentaho Data Integration, permite la creación de dos tipos de procesos: transformaciones (*transformations*), y trabajos (*jobs*). La principal diferencia está en la forma de ejecución: mientras que las transformaciones ejecutan de forma concurrente todos los objetos que las componen, los *jobs* son ejecutados de forma secuencial. Por este motivo, las primeras suelen ser empleadas para gestionar el flujo de datos, mientras que los segundos tienden a emplearse para manejar el flujo de ejecución, llamando a unas transformaciones u otras según se cumplan o no ciertas condiciones.

Además, es posible realizar procesos más generales ya que se soportan argumentos, parámetros y variables. Así, es posible reutilizar una transformación o un *job* para realizar distintas funciones.

8.2. Instalación y configuración de Pentaho Data Integration

La versión libre de Pentaho puede descargarse de forma gratuita desde la dirección <http://community.pentaho.com>. Está escrito en Java, por lo que para comenzar a trabajar basta con descomprimir el fichero zip descargado. En Windows, es necesario crear una variable de entorno PENTAHO_JAVA_HOME que incluya la ruta de Java en el equipo, mientras que en Linux debe añadirse la línea `export PENTAHO_JAVA_HOME=[ruta java]` al archivo `/etc/environment`.

Los tres componentes principales pueden ejecutarse utilizando los *scripts* incluidos en la carpeta raíz:

- `spoon.bat`: interfaz gráfica para el diseño y ejecución de transformaciones y *jobs*.
- `pan.bat`: interfaz por línea de comandos para la ejecución de transformaciones.
- `kitchen.bat`: interfaz por línea de comandos para la ejecución de *jobs*.

Pentaho puede leer y escribir archivos en disco, o trabajar con un repositorio de base de datos, ofreciendo la posibilidad de que varios desarrolladores puedan almacenar su trabajo en un lugar común. Su configuración es posible desde Spoon. Por ejemplo, de cara a configurar un repositorio local de desarrollo se deben seguir los pasos detallados a continuación.

Como se ve en la figura 8.1, al abrir `spoon.cmd` aparece una ventana de selección de repositorio. Haciendo clic en el símbolo ⊕, el programa solicita el tipo de repositorio a crear.

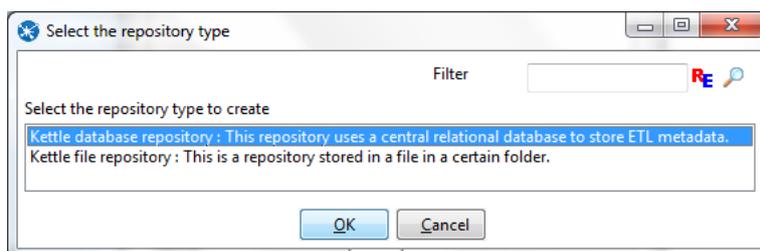


Figura 8.1: Selección de tipo de repositorio.

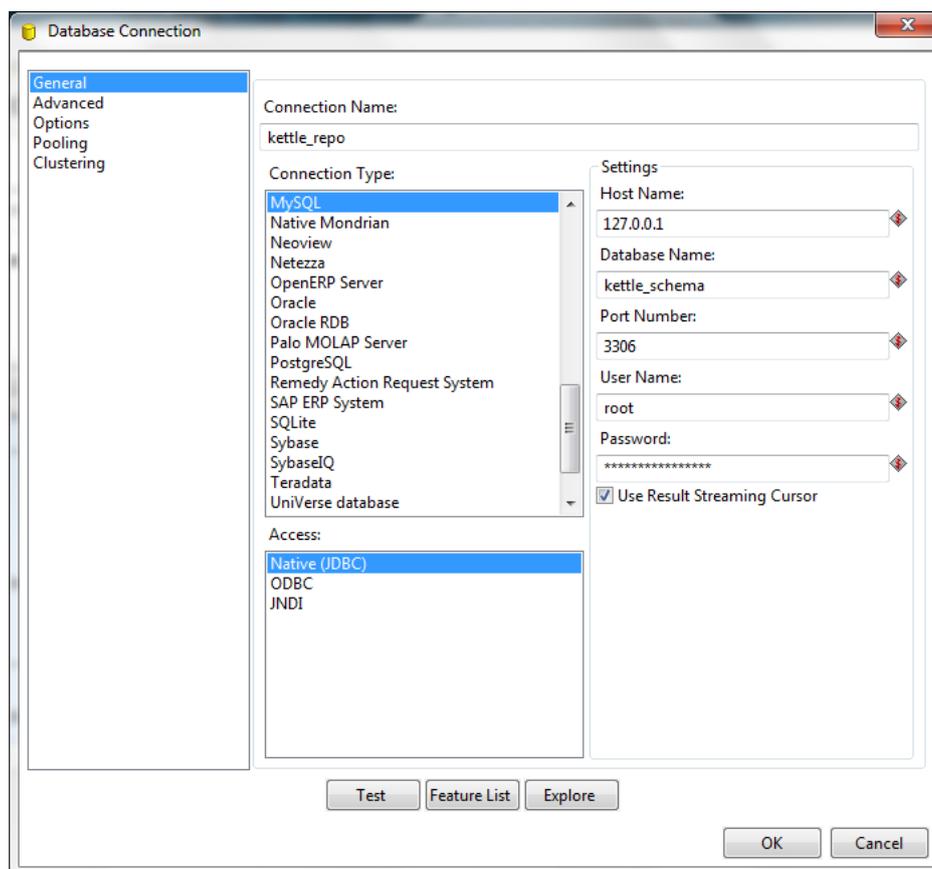


Figura 8.2: Selección de base de datos.

A continuación, como se muestra en la figura 8.2, debe definirse el servidor de base de datos a utilizar. En este caso, se ha creado un *schema kettle_schema* en un servidor local de MySQL. Como el servidor es local, podemos utilizar el usuario *root*, si bien esta práctica no es aconsejable en un servidor productivo. En ese entorno se ha utilizado un usuario *kettle*, cuyos permisos se han configurado específicamente para este fin.

Una de las grandes bazas de Pentaho es que es muy configurable, y soporta numerosos RDBMS, a costa de aumentar ligeramente la complejidad de la puesta en marcha. En concreto, para interactuar con un servidor MySQL es necesario realizar algunos ajustes más finos. En particular, como se ve en la figura 8.3, para la versión de MySQL empleada debe desactivarse la opción *Supports the boolean data type*, que viene marcada por defecto.¹

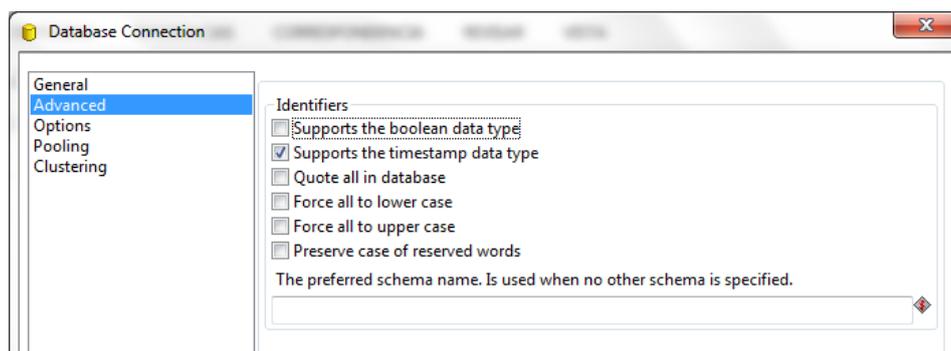


Figura 8.3: Parámetros avanzados del servidor.

Una vez se han definido los datos de conexión, cuya corrección puede comprobarse mediante el botón *Test*, aparece la ventana de la figura 8.4.

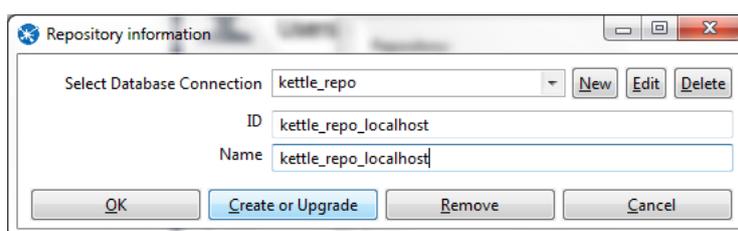


Figura 8.4: Definición del repositorio.

Spoon procederá a crear las tablas necesarias en base de datos al pulsar el botón *Create or Upgrade* y mostrará el resultado en una ventana similar a la de la figura 8.5.

¹Esta incidencia fue resuelta en el JIRA PDI-11617: <http://jira.pentaho.com/browse/PDI-11617>.

```

Results of the SQL statements

The SQL statements had the following results

SQL executed: CREATE TABLE R_REPOSITORY_LOG
(
  ID_REPOSITORY_LOG BIGINT NOT NULL PRIMARY KEY
  , REP_VERSION VARCHAR(255)
  , LOG_DATE DATETIME
  , LOG_USER VARCHAR(255)
  , OPERATION_DESC MEDIUMTEXT
)
SQL executed: CREATE TABLE R_VERSION
(
  ID_VERSION BIGINT NOT NULL PRIMARY KEY
  , MAJOR_VERSION INT
  , MINOR_VERSION INT
  , UPGRADE_DATE DATETIME
  , IS_UPGRADE CHAR(1)
)
SQL executed: INSERT INTO R_VERSION(ID_VERSION, MAJOR_VERSION, MINOR_VERSION, UPGRADE_DATE, IS_UPGRADE)
SQL executed: CREATE TABLE R_DATABASE_TYPE
(
  ID_DATABASE_TYPE BIGINT NOT NULL PRIMARY KEY
  , CODE VARCHAR(255)
  , DESCRIPTION VARCHAR(255)
)
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (1, 'DERBY', 'Apache
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (2, 'AS/400', 'AS/40
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (3, 'INTERBASE', 'Bo
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (4, 'INFINIDB', 'Cal
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (5, 'DBASE', 'dBase
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (6, 'EXASOL4', 'Exas
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (7, 'EXTENDB', 'Exte
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (8, 'FIREBIRD', 'Fir
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (9, 'GENERIC', 'Gene
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (10, 'GREENPLUM', 'G
SQL executed: INSERT INTO R_DATABASE_TYPE(ID_DATABASE_TYPE, CODE, DESCRIPTION) VALUES (11, 'SOBASE', 'Gue

```

Figura 8.5: Las tablas del repositorio han sido creadas correctamente.

Tras su correcta creación, se puede comenzar a trabajar utilizando el repositorio. El último paso previo al comienzo de la programación de procesos ETL es definir las bases de datos a emplear como entrada y/o salida. Esto se puede realizar desde el menú mostrado en la figura 8.6.

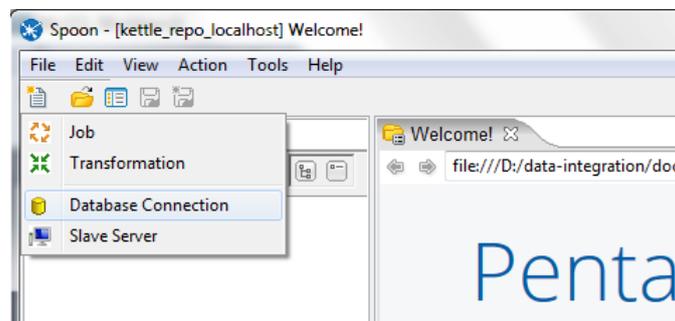


Figura 8.6: Creación de nueva conexión a base de datos.

Procediendo a configurarla tal y como se hizo al definir el repositorio desde una ventana idéntica a la de la figura 8.2.

Siguiendo este mismo procedimiento, se ha creado otro repositorio en un esquema denominado `rm_kettle` en la base de datos de producción. Por tanto, los procesos ETL serán desarrollados y testados en el entorno local, para ser subidos a producción una vez su funcionamiento haya sido validado.

8.3. Implementación del ETL

Tras crear las tablas especificadas en el capítulo 7, se han creado los procesos ETL que se encargan de rellenarlas. El primer conjunto se ocupa de realizar las cargas en el NDS después

de realizar una pequeña limpieza de los datos obtenidos de las bases de datos operacionales, mientras que el segundo lee información de éste y la cargará, tras realizar las transformaciones oportunas, en el DDS.

8.3.1. NDS

8.3.1.1. cliente_datos

8.3.1.1.1 Job

El objetivo del *job* `cliente_datos` es llamar a las transformaciones necesarias para realizar la carga de las tablas de tipo `cliente_`.

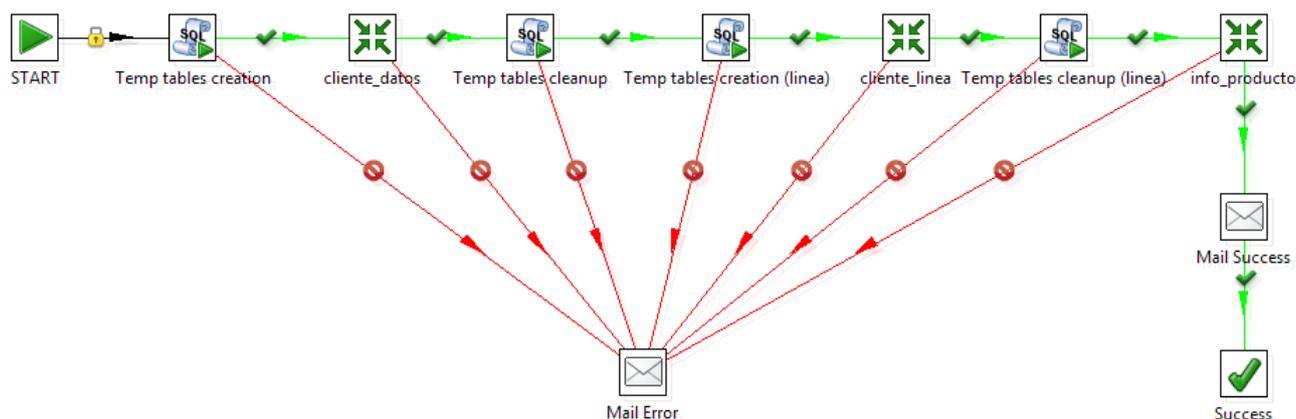


Figura 8.7: *Job* `cliente_datos`

Todo *job* consiste en una serie de pasos ejecutados secuencialmente. El único objeto que debe contener obligatoriamente un *job* es *START*, que marca el comienzo del proceso. Por otro lado, se recomienda la utilización de dos objetos de tipo *Mail*, usándose para enviar un correo electrónico al finalizar la ejecución, ya sea ésta correcta o incorrecta. Así, se tendrá la certeza de que el proceso ha tenido lugar, ya que la práctica más habitual es que se programen de tal forma que se ejecuten en el momento en que la base de datos pase por un período de mínima carga (de madrugada). Estos correos contendrán los *logs* de todas las transformaciones, así como algunos archivos útiles generados durante ésta, como la relación de los registros que no han pasado cierta comprobación de calidad, o que presentaban descuadres.

Además de las transformaciones a las que se va llamando, el *job* `cliente_datos` cuenta con cuatro pasos *Execute SQL script*. Dos de ellos utilizan sentencias `CREATE TABLE AS SELECT` para crear tablas temporales que ayudan a reducir en gran medida el tiempo necesario para la ejecución. Estas tablas contienen datos filtrados e indexados, posibilitando su uso posterior, y evitando el uso de *sub-queries*, con la consiguiente mejora en rendimiento. Los otros dos pasos ejecutan sentencias `DROP` para ahorrar espacio entre ejecuciones.

8.3.1.1.2 Transformación `cliente_datos`

El objetivo de esta transformación, mostrada en la figura 8.8, es integrar los datos de los clientes que hay almacenados en el sistema del MVNE (`rm_mvno`) y en la base de datos del portal

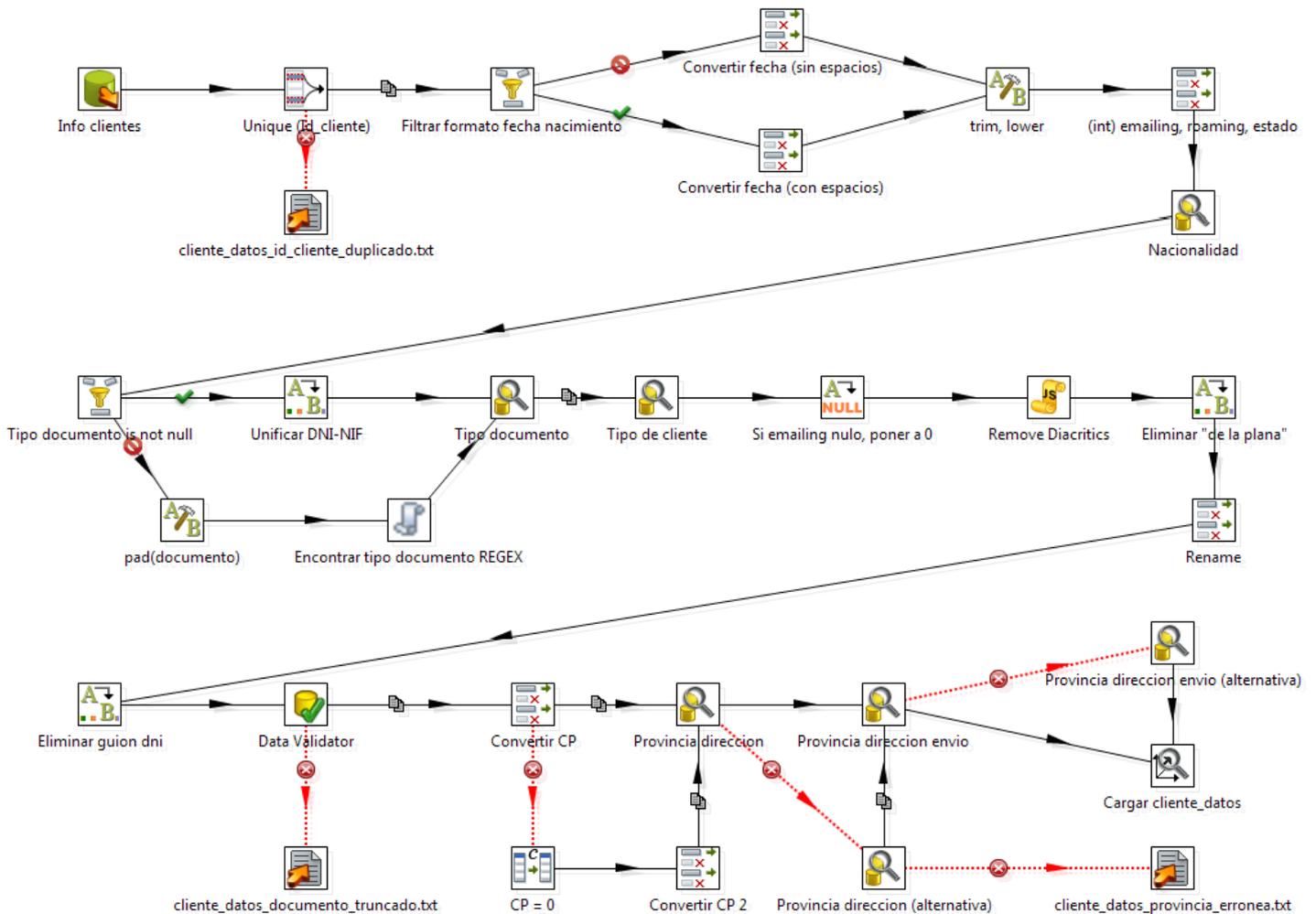


Figura 8.8: Transformación cliente_datos

web (rm_walva). Se asume que los datos del MVNE son los oficiales, por lo que los datos del portal suelen emplearse para rellenar los registros incompletos.

La transformación comienza con un paso *Table input* que lee información de las bases de datos productivas, ya sea directamente o a través de las tablas temporales que fueron creadas anteriormente en el *job*. Esta consulta SQL debería producir una única fila por *Id_cliente*, por lo que inmediatamente después se detectan los posibles duplicados mediante un paso *Unique rows*, quedando descartados del flujo del ETL y almacenándose en un fichero de texto para posterior análisis.

A continuación, se convierten los datos que *Table input* no ha podido interpretar. En concreto, la fecha de nacimiento se almacena en base de datos como un *string*. Dado que algunos registros pasados no siguen el mismo formato que los actuales, es necesario utilizar *Filter rows* para alimentar los pasos *Select/Rename values* que convierten la fecha a tipo *Date*. Después de emplear *String Operations* para eliminar espacios innecesarios y utilizar minúsculas en todas las cadenas de texto, se utiliza de nuevo *Select/Rename values* para convertir los flags de estado de *emailing*, *roaming* y cliente a entero.

La mayoría de los datos que deben ser normalizados en esta fase aparecen como texto en la base de datos operacional. Para encontrar el identificador asociado en la tabla que almacena los

valores posibles, se utiliza *Database Value Lookup*, que permite devolver la clave primaria del valor encontrado en la tabla. Antes de realizar esta operación para el campo **Tipo Documento**, se emplea *Replace in string* para sustituir *DNI* por *NIF* en aquellos registros que usan notación antigua, y se examina el campo **Documento** para completarlo en caso de que esté vacío. Esta función la cumple una *User defined Java class* definida por el código incluido en el Anexo A.1. Otra función personalizada empleada es la utilizada en el paso **Remove diacritics**, de tipo *Modified Java Script Value*, que se encarga de eliminar tildes (por ejemplo, pasar de “ó” a “o”).

Siguiendo con la uniformización de datos entre distintos sistemas y versiones de éstos, un objeto de tipo *If field value is null* da por defecto valor 0 al campo **emailing**, que indica si el cliente está dispuesto a recibir comunicaciones por correo electrónico. Asimismo, se hace que *Castellón* y *Castellón de la plana* sean la misma provincia, y se elimina el guión que aparece en algunos documentos de identidad. La longitud de este dato se comprueba inmediatamente a continuación en el paso *Data validator*.

A continuación, se trata de convertir el código postal de las direcciones de envío y facturación a entero. De no ser posible, se hace que su valor sea cero. Para normalizar el último campo, se hace *lookup* de la provincia de ambas direcciones, siendo necesario hacer búsqueda en dos ocasiones en la tabla **info_provincia**, ya que cada columna utiliza la nomenclatura de un sistema.

Una vez se han realizado todas las transformaciones pertinentes, el último paso es realizar la carga en la tabla **cliente_datos**. Dado que entre los requerimientos está el de mantener un histórico de los datos, se utiliza el paso *Dimension Lookup/Update*. Este paso, cuya configuración se muestra en la figura 8.9, busca en la tabla destino la última versión del registro cuya clave coincide con los campos especificados en la pestaña *Keys* y compara los indicados en la pestaña *Fields*, realizando la acción escogida en la columna *Type of dimension update*. Entre las opciones disponibles destacan:

- *Insert*, que añade una nueva fila con el campo actualizado.
- *Punch through*, que actualiza todos los registros históricos al nuevo valor del campo.
- *Update*, que actualiza el último registro al nuevo valor del campo.

Este paso gestiona automáticamente la clave técnica (que hace de clave primaria en la tabla, en contraposición a la clave natural, que sería la composición de **Id_cliente** y **version**), si bien en este caso se ha optado por dejar que sea el RDBMS el que lo gestione mediante una clave primaria con *auto increment*, las fechas de validez de cada registro mediante los campos **date_from** y **date_to** y la versión del registro, mediante el campo **version**. Aunque el uso de la caché puede reducir en gran medida el tiempo de carga, no es útil en este caso, ya que la clave **Id_cliente** aparece una única vez por ejecución.

8.3.1.1.3 Transformación **cliente_linea**

La transformación **cliente_linea** tiene los mismos objetivos que **cliente_datos**, con la salvedad de que maneja datos sobre líneas. El proceso se muestra en la figura 8.10.

La fuente de datos principal es la tabla **rm_msisdn**, que va uniéndose mediante **left join** a las tablas de **rm_valva** mediante tres caminos distintos, posibilitando encontrar la mayor parte de datos. Asimismo, se utiliza **inner join** con **cliente_datos** para garantizar que únicamente los registros cuyo **Id_cliente** está en esta tabla se consideren.

Una vez se han leído los datos, se comienza con las validaciones de unicidad y formato del **MSISDN**, la conversión a formato nativo de fechas y enteros –que fallará cuando la fecha de

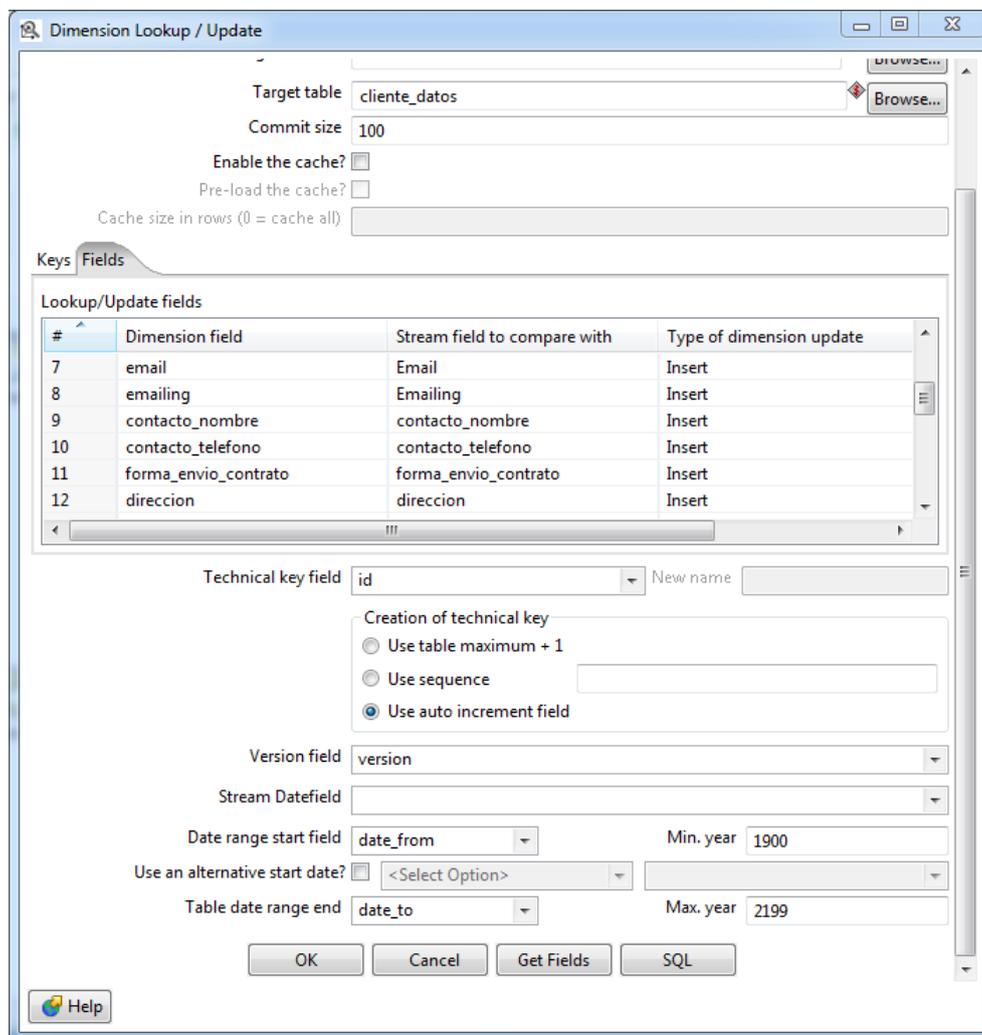


Figura 8.9: *Dimension lookup/update*

portabilidad esté incompleta, indicando que el número todavía no ha portado, posponiéndose su carga-, y comprobando el formato del MSISDN del padrino. Cuando el campo contiene un valor no numérico, se interpreta como un error de imputación de datos en el momento del alta, y el valor pasa a ser el del *sponsor provider*.

A continuación, se comienza con la normalización. Al contrario que en la transformación anterior, las tablas donde se realiza la búsqueda no son estáticas, sino que nuevas líneas de negocio generan nuevos valores. Por tanto, es necesario buscar el valor, e insertarlo de no existir. Para esto es posible emplear *Combination lookup/update*. La diferencia de este paso con *Database lookup* es que éste último no actualiza la tabla, sólo lee valores ya existentes.

Hay dos puntos en los que los registros toman distintos flujos según cumplan o no ciertas condiciones mediante la utilización de *Filter rows*. En primer lugar, se utiliza para emular el comportamiento del paso *Combination lookup/update* para los casos en los que el campo *sim_type* llega vacío. Esto es necesario para que el campo *tipo_sim* pueda tomar valor 0, ya que la ausencia de este campo no significa que el alta fuera incorrecta, sino que se realizó a través del CRM, que no almacena esta información.

En segundo lugar, *Filter rows* se emplea para procesar las portabilidades de distinta forma que las nuevas altas, ya que las primeras deben manejar también información sobre el IC-

CID de origen y el operador donante, que debe ser tomado de la información disponible en `rm_mvno.rm_alta_portabilidad` salvo cuando se trata de portabilidades internas, que debe tomarse de `rm_walva.numberPortDetailData`. Se valida la longitud del ICCID utilizando un paso *User Defined Java Expression* que emplea el siguiente código.

Código 8.1: *User defined java expression* Longitud ICCID 19/20

```
1 ICCID_origen.length() == 20 || ICCID_origen.length() == 19
```

Cada registro debe llegar al paso *Carga* con el mismo número de campos, sin importar el camino a través del cuál lo hace. Para ello, se añade el campo `info_operador_origen_id` en aquellas vías de ejecución en las que no se utiliza un paso de *lookup* para encontrarlo, mediante los pasos *Nuevo campo Operador donante* e `info_operador_origen_id = 0`.

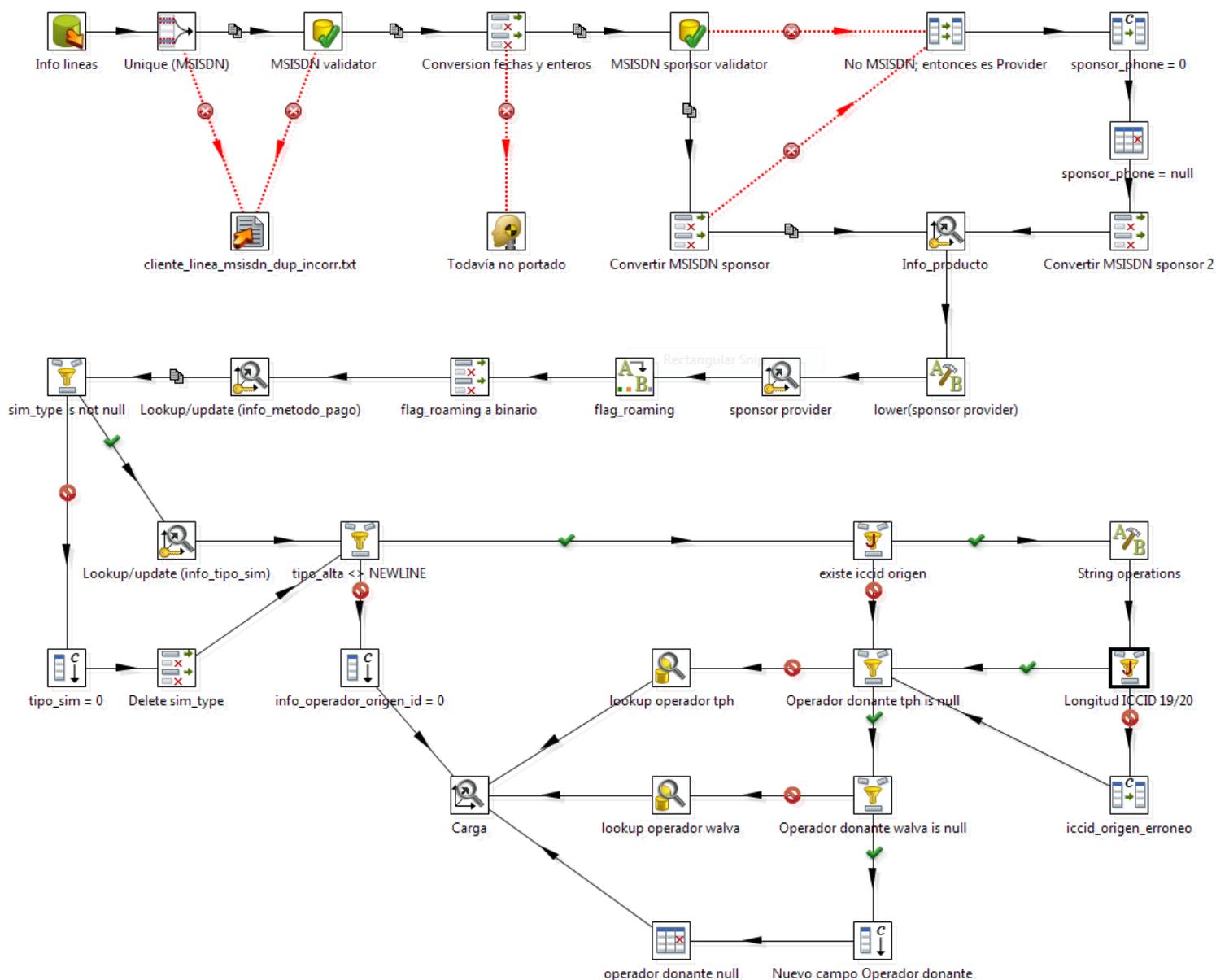


Figura 8.10: Transformación `cliente_linea`

8.3.1.1.4 Transformación info_producto

En la figura 8.11 se muestra la última transformación ejecutada en el *job*. La transformación anterior, *cliente_linea*, inserta un registro en la tabla *info_producto* por cada nueva tarifa que se encuentra durante su ejecución, rellenando únicamente un identificador autonumérico y el nombre. El proceso actual se encarga de leer los demás datos del producto de la tabla *rm_walva.Rate* y actualizar *rm_bi_nds.info_producto* con esta información. También se actualizarán los datos de tarifas existentes anteriormente, en caso de haber modificado su precio o alguna de sus características.

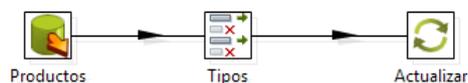


Figura 8.11: Transformación info_producto

8.3.1.2. cliente_pioneros

8.3.1.2.1 Job

Al contrario que el *job* *cliente_datos*, *cliente_pioneros* (figura 8.12) está diseñado para ser ejecutado de forma manual cuando sea necesario. En concreto, debe ejecutarse después de realizar los cálculos mensuales del Plan Pioneros, ya que debe almacenar información de saldos y tarjetas monedero. Por ello, no es necesario incluir pasos que envíen el resultado por correo electrónico.



Figura 8.12: Job cliente_pioneros

Otra peculiaridad de este *job* es que permite escoger el primer período a considerar mediante un argumento especificado en el momento de la ejecución ya sea por línea de comandos o a través de la GUI.

8.3.1.2.2 Transformación cliente_pioneros_tablas

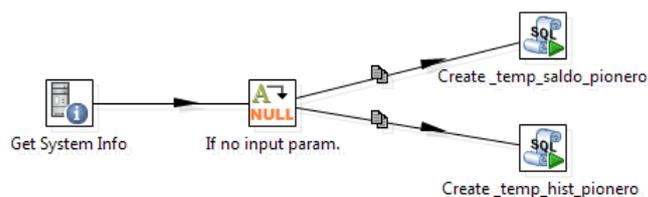


Figura 8.13: Transformación cliente_pioneros_tablas

La primera transformación, mostrada en la figura 8.13, se encarga precisamente de leer el argumento mediante el paso *Get system info*, que genera una fila con un campo por cada argumento de entrada. Con un objeto *Replace null value* se hace que la fecha por defecto sea 1900-01-01 00:00:00, con lo que se cargarán todos los datos históricos en caso de que no se introduzca ningún valor. Por último, se crean tablas temporales a partir de las tablas de tipo *mgm* almacenadas en *rm_walva*, escogiendo los registros con cláusulas como:

```
1 where date(concat(mgmh_fecha_periodo, '01')) >= ?
```

donde Kettle reemplazará automáticamente el carácter ? por la fecha introducida en el argumento. De esta forma, los datos quedan filtrados para las siguientes transformaciones, ganando en rendimiento.

8.3.1.2.3 Transformación cliente_pioneros

La transformación *cliente_pioneros* (figura 8.14) toma los datos de la tabla *_temp_saldo_pionero*, directamente ordenados según su *Id_cliente* y *Fecha_periodo*. Dado que los datos ya han sido filtrados previamente, solo se realizan las conversiones de tipo de datos necesarias antes de cargar, manteniendo un histórico con *Dimension lookup/update*.

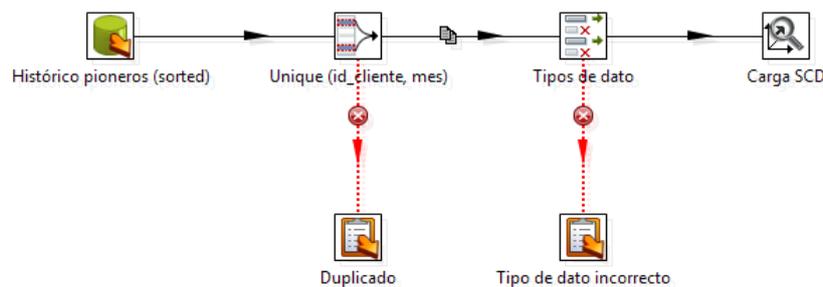


Figura 8.14: Transformación *cliente_pioneros*

8.3.1.2.4 Transformación cliente_pioneros_monedero

La transformación *cliente_pioneros_monedero* (figura 8.15) lleva a cabo la carga de los datos de saldo y tarjetas monedero de forma similar.

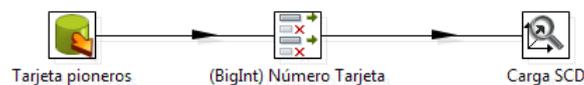


Figura 8.15: Transformación *cliente_pioneros_monedero*

8.3.1.3. cdr_carga_inicial

8.3.1.3.1 Job

El proyecto de implementación del DWH fue concebido meses después de la salida al mercado del operador. Por tanto, es necesario disponer de mecanismos para realizar la carga de datos

pasados de forma eficiente. En concreto, las tablas de CDRs son las de mayor volumen. Esto hace que la utilización de pasos que comprueben la existencia de cada registro a insertar entre los anteriores sean inutilizables.

Por ello, se ha dividido el proceso en dos partes. En primer lugar, se realiza una carga masiva única con todos los datos disponibles hasta el momento de la ejecución. Para esto, en el *job cdr_carga_inicial*, mostrado en la figura 8.16, se utilizan las tablas `t_cdr` de `rm_walva`, que se utilizaban anteriormente para almacenar los CDRs mensuales frente a las tablas diarias de `rm_mcdf`. Posteriormente, se realizan cargas incrementales utilizando el *job cdr_carga_incremental*.

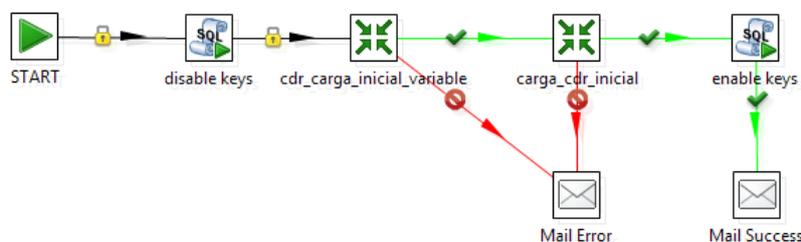


Figura 8.16: *Job cdr_carga_inicial*

Como se comentaba en la sección 2.3.4, es útil desactivar los índices durante cargas masivas para ahorrar tiempo de CPU y mantener el árbol de índices compacto y desfragmentado. De esto se encargan los pasos `Disable keys` y `Enable keys`, situados al principio y al final del *job*.

Por otra parte, dado que la lectura se realiza desde tablas mensuales, el proceso recibe el mes a cargar como argumento. El modo de realizar la carga inicial completo es, por tanto, llamar al *job* tantas veces como sea necesario utilizando un *bash script*. Para facilitar el análisis de los *logs*, se envían los resultados por correo electrónico.

8.3.1.3.2 Transformación `cdr_carga_inicial_variable`

Esta transformación, mostrada en la figura 8.17, se encarga de leer el argumento de entrada y guardar su valor en la variable `YEARMONTH`, para la cuál se define un valor por defecto previo al primer mes con datos, y un *scope* tal que sea válida en todo el *job* padre. De este modo podrá ser leída por la siguiente transformación.



Figura 8.17: Transformación `cdr_carga_inicial_variable`

Este método más complejo es necesario porque la variable va a formar parte del nombre de la tabla a leer por un paso *Table input*.

8.3.1.3.3 Transformación `carga_cdr_inicial`

Como se ve en la figura 8.18, una vez que la variable `YEARMONTH` tiene datos, Kettle procede a sustituirla automáticamente en el paso *Table input*, siempre y cuando la opción *Replace variables in script?* esté activada.



Figura 8.18: Transformación carga_cdr_inicial

La única modificación que deben sufrir los datos es la normalización de cuatro de sus campos mediante pasos *Combination lookup/update*, la sustitución de valores nulos en las zonas de roaming por el valor 0, y la conversión de las cadenas de texto *Fecha* y *Hora* a una única variable de tipo *Datetime*. Por último, se cargan todos los datos en la tabla *cdr_historico* mediante un paso *Table output*.

8.3.1.4. cdr_carga_incremental

8.3.1.4.1 Job

Tras la carga inicial de CDRs, es necesario añadir los nuevos registros diariamente. Dado el gran volumen de datos, no es viable analizar cada registro uno a uno. Aprovechando el hecho de que el MVNE únicamente realiza cargas de nuevos registros –no actualiza ni elimina–, en el *job cdr_carga_incremental* (figura 8.19) sólo se buscarán nuevos registros en las tablas de CDRs diarias (de *rm_mcdf*) cuando el número de filas sea distinto al número de registros para ese mismo día en la tabla *cdr_historico*. La transformación auxiliar *cdr_carga_incremental_filas* debe ser ejecutada antes que este *job* de forma que el archivo generado por ésta, que contiene las fechas con diferencias en número de registros, sea leído por un *script* que llame al *job* tantas veces como sea necesario, introduciendo la fecha a procesar como parámetro. Dado que los CDRs generados por líneas en *roaming* tardan varios días en recibirse, no es suficiente con cargar únicamente los CDRs de cada día.

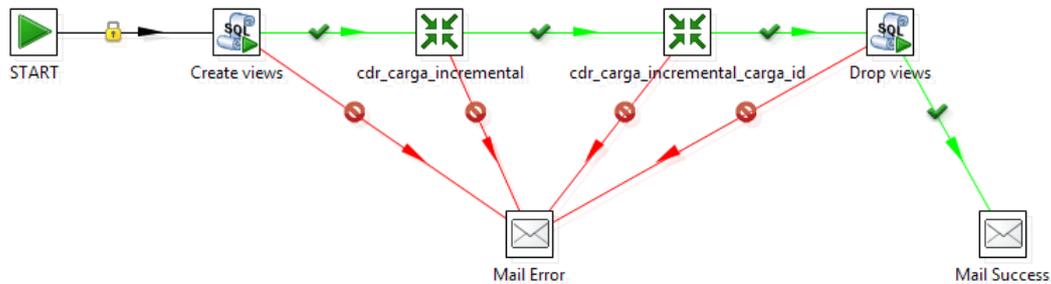


Figura 8.19: Job cdr_carga_incremental

En realidad, el *job* recibe tres parámetros: las fechas de inicio y fin, y la misma fecha en el formato del nombre de la tabla diaria. Los tres son calculados en el *script* que lanza su ejecución sobre *kitchen.sh*, introduciéndolos por línea de comandos para su utilización en el paso *Create views*, que crea dos vistas: una que apunta a los registros de *cdr_historico* entre las dos fechas, y otra que apunta a todos los registros de la tabla *rm_mcdf.rm_cdrs\${VAR_TABLE_DATE}*. Después, la transformación *cdr_carga_incremental* compara esas vistas, para pasar a realizar las cargas necesarias en *cdr_carga_incremental_carga_id*.

8.3.1.4.2 Transformación auxiliar cdr_carga_incremental_filas

El objetivo de esta transformación, mostrada en la figura 8.20, es comparar el número de filas de cada una de las tablas diarias de CDRs de `rm_mcdf` con el número de registros por cada día en la tabla `cdr_historico` del NDS, generando un fichero que contenga los días para los que el `cdr_historico` no contiene toda la información.

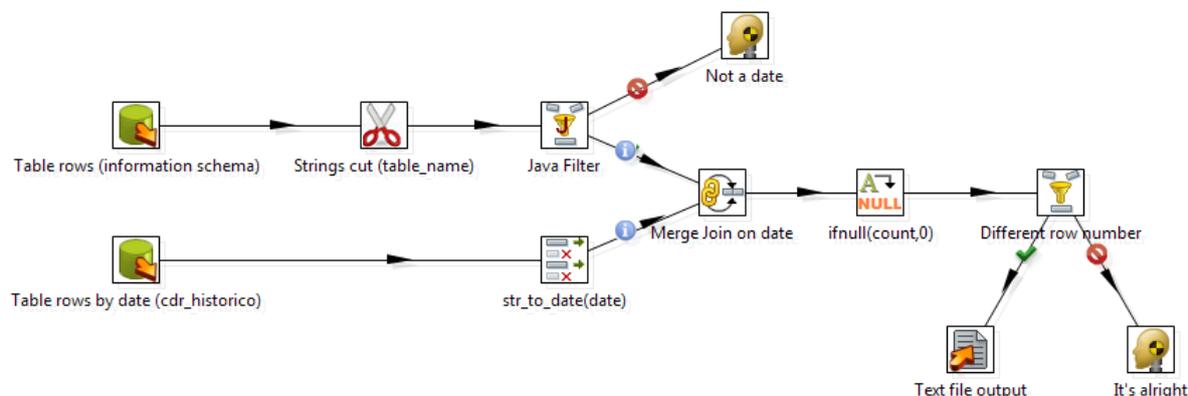


Figura 8.20: Transformación `cdr_carga_incremental_filas`

La primera variable puede obtenerse de forma sencilla utilizando el *Information Schema* de MySQL, empleando la consulta:

Código 8.2: *Table input* Table rows (information schema)

```
1 SELECT table_name, TABLE_ROWS
2 FROM INFORMATION_SCHEMA.TABLES
3 WHERE TABLE_SCHEMA = 'rm_mcdf'
4 ORDER BY table_name
```

El proceso continúa pasando el resultado de esta consulta a `Strings cut (table_name)`, que conserva la parte del nombre de la tabla que indica la fecha, en formato `YYYYMMDD`. Dado que también hay tablas mensuales cuyo formato es `YYYYMM`, el paso `Java filter` posterior únicamente permite el paso de aquellas filas en las que la longitud de la variable sea 8.

Por otro lado, la segunda variable a comparar se puede obtener con la consulta:

Código 8.3: *Table input* Table rows by date (cdr_historico)

```
1 select
2 date(Fecha),
3 count(*)
4 from rm_bi_nds.cdr_historico
5 group by date(Fecha)
6 order by date(Fecha)
```

La variable `Fecha` se convierte a tipo fecha en formato `YYYYMMDD` y se une a la fila correspondiente de la otra rama mediante un `LEFT JOIN` en el que la variable principal es la que viene del *information schema*. A los valores nulos se les asigna el valor 0, y se procede a aplicar el filtro `Different row number`, guardándose aquellos que tengan distinto número de días en el archivo `temp_filas`.

8.3.1.4.3 Transformación cdr_carga_incremental

El objetivo de esta transformación (figura 8.21) es localizar las filas que deben ser actualizadas. En primer lugar, se leen las vistas creadas en el *job* mediante los pasos *Daily CDRs* y *CDR hist*. El encargado de realizar la comparación es el paso *Merge rows (diff)*, que debe recibir exactamente las mismas columnas por cada una de sus entradas, ambas ordenadas por la clave por la que va a comparar. En este caso, utilizando *Id_llamada*, buscará diferencias entre las dos vistas, creando un campo *flag* adicional y marcando cada fila como *identical*, *changed*, *new* o *deleted*. El uso de este proceso supone una mejora de rendimiento frente a *Dimension lookup/update*.

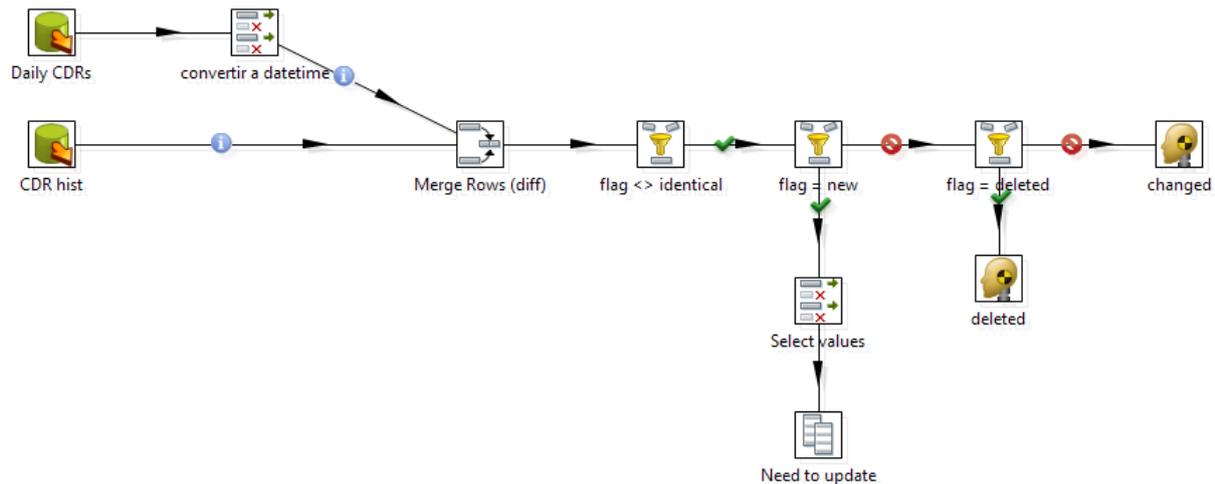


Figura 8.21: Transformación cdr_carga_incremental

Mediante diversos filtros se redirigen las filas de acuerdo con su estado. En este caso sólo se han considerado las filas nuevas, tomando su *Id_llamada* en el paso *Select values*. Utilizando *Need to update*, paso del tipo *Copy rows to result*, se pasan todas las filas al resultado de forma que puedan ser leídas por la siguiente transformación.

8.3.1.4.4 Transformación cdr_carga_incremental_carga_id

Como se ve en la figura 8.22, la información que ha pasado la transformación anterior se recibe con un paso *Get rows from result*. Utilizando *Table input*, con las opciones *Replace variables in script* y *Execute for each row* activadas, es posible lanzar una consulta para cada uno de los *Id_llamada* a actualizar desde la vista creada previamente en el *job*:

Código 8.4: *Table input* Carga diaria CDRs

```
1 SELECT * from 'rm_bi_nds'._temp_cdr_tabla_fecha
2 where Id_llamada = ?
```

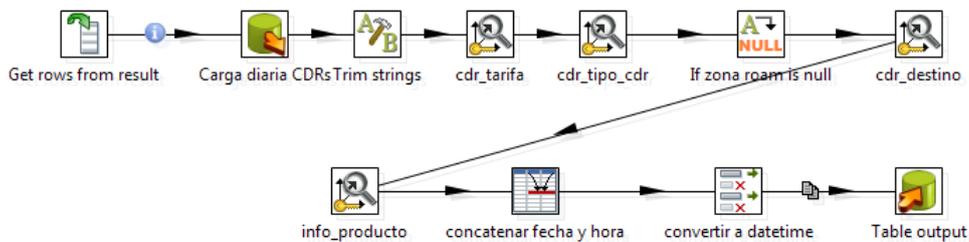


Figura 8.22: Transformación `cdr_carga_incremental_carga_id`

Los demás pasos son exactamente iguales a los de `cdr_carga_inicial`.

8.3.1.5. factura

8.3.1.5.1 Job

El *job factura* (figura 8.23) controla todo el proceso de carga de las tablas relacionadas con facturación. La única transformación que requiere tablas temporales es `factura_cliente`, creándose éstas antes de su ejecución, y eliminándose inmediatamente después.

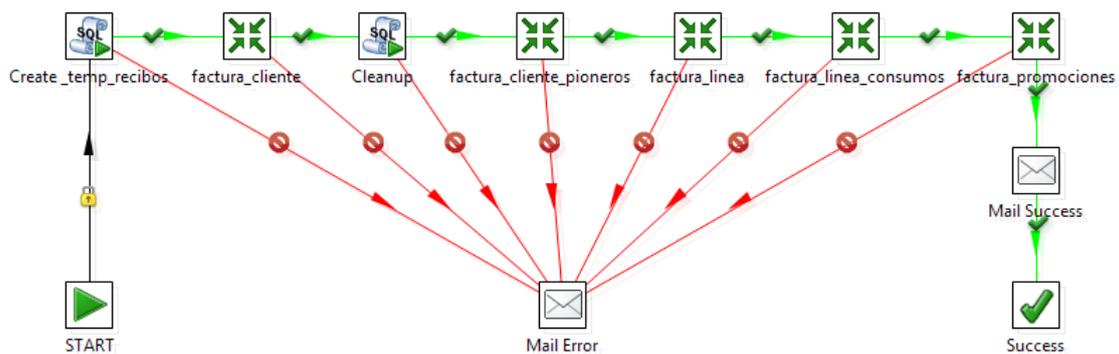


Figura 8.23: Job factura

8.3.1.5.2 Transformación `factura_cliente`

La transformación (figura 8.31) lee datos de la tabla `fac_cli_datos_unicos`, utilizando `INNER JOIN` con `cliente_datos`, para evitar incluir facturas de clientes que por alguna razón aún no están en el NDS, y con la tabla temporal `_temp_recibos` creada en el *job*. La creación de esta tabla resulta necesaria porque la tabla `rm_xfac.Recibos` contiene todos los cobros que el MVNE ha tratado de realizar junto a su estado: (0: Pendiente de pago; 1: Cobrado; 2: Devuelto).

Esto resulta problemático en nuestro caso ya que una factura con un importe de 5 euros podía aparecer en dos filas, una de 4 euros en estado *cobrado*, y una de 1 euro marcada como *pendiente de pago*. Para consolidar la información, se decidió seguir la siguiente lógica:

- Si alguno de los recibos está pendiente, el Estado es 0.
- Si el importe pagado es el mismo que el importe del recibo, el Estado es 1.

- Si no, el Estado estado es 2.

Esto se implementó mediante la siguiente consulta SQL, junto a un nuevo campo que recogiera el importe que el operador había cobrado en realidad:

Código 8.5: Sentencia CREATE de la tabla `_temp_recibos`

```

1 CREATE TABLE IF NOT EXISTS rm_bi_nds._temp_recibos
2 (PRIMARY KEY (NumFact))
3 ENGINE = MEMORY
4 select NumFact, importe_recibo, importe_recibo_cobrado, Cliente,
5 if(importe_recibo_cobrado = importe_recibo, 1,
6     if(min(Estado)=0, 0, 2)
7 ) as Estado
8 from
9 (
10     select trim(NumFact) as NumFact,
11         sum(Importe) as importe_recibo,
12         sum(case when Estado = 1 then Importe else 0 end) as
13             importe_recibo_cobrado,
14         Estado,
15         trim(Cliente) as Cliente
16     from 'rm_xfac'.Recibos
17     group by trim(NumFact)
18 ) as r1
19 group by NumFact;

```

Tras la lectura de información de la base de datos, la transformación unifica tipos de datos de las fuentes empleadas, aplica un mismo formato a todas las cadenas de texto, y realiza diversas validaciones. En primer lugar, el filtro `Check id_cliente` comprueba que el `id_cliente` de la tabla `cliente_datos` y el indicado en la tabla de recibos es el mismo, ya que podría haber quedado desactualizado tras un cambio de cliente. Esto es necesario porque la sentencia `JOIN` se hace a través del número de factura.

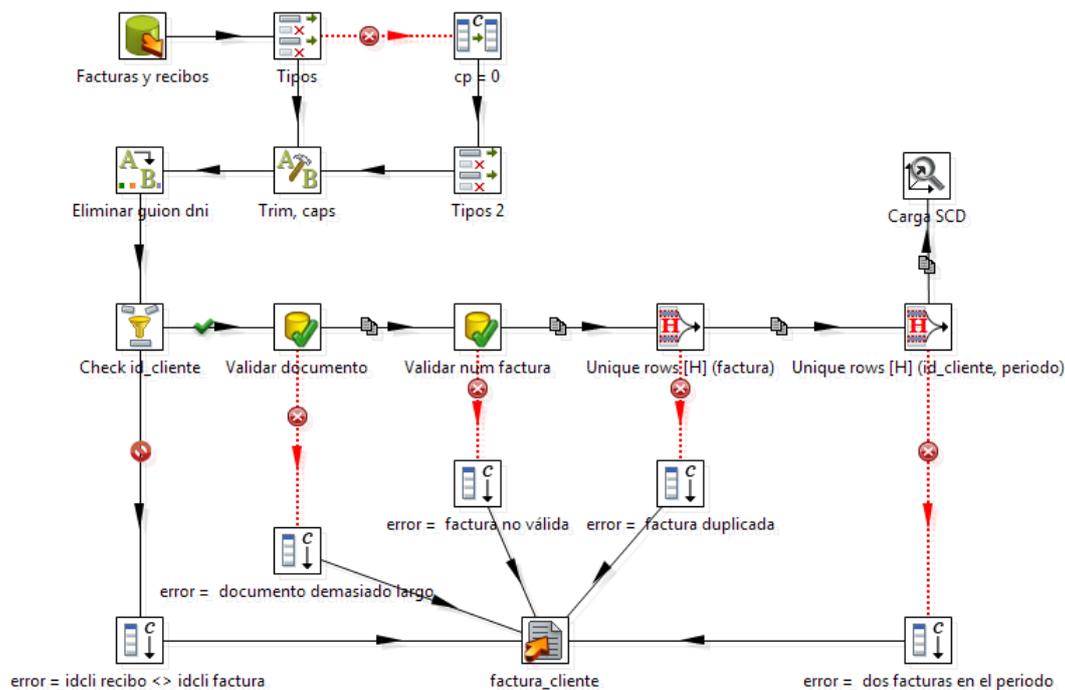


Figura 8.24: Transformación `factura_cliente`

En segundo lugar, se validan la longitud del documento de identidad y el formato del número de factura mediante expresiones regulares. Por último, empleando pasos *Unique rows (Hashset)*, se asegura que todos los registros cargados son únicos. En concreto, debe haber una única fila por número de factura y, a su vez, una única factura por cliente y período de facturación. La ventaja de este paso frente a *Unique rows* es que no hace falta que la entrada esté ordenada según la clave cuya unicidad quiere garantizarse. Esto es especialmente útil en este caso, dado que quieren evaluarse dos claves.

En transformaciones anteriores se guardaba un archivo por cada posible tipo de error en los registros. En este caso se guarda un único archivo, añadiendo una columna que especifica la clase de error detectada.

8.3.1.5.3 Transformación `factura_cliente_pioneros`

Como se muestra en la figura 8.25, los datos de los descuentos aplicados por el Plan Pioneros a cada factura se leen de la tabla `mgm_datos_mes_factura_pioneros`. Mediante el `INNER JOIN` con la tabla `factura_cliente` que tiene lugar en el paso *Merge join* se garantiza que la factura ya haya sido guardada en el NDS.

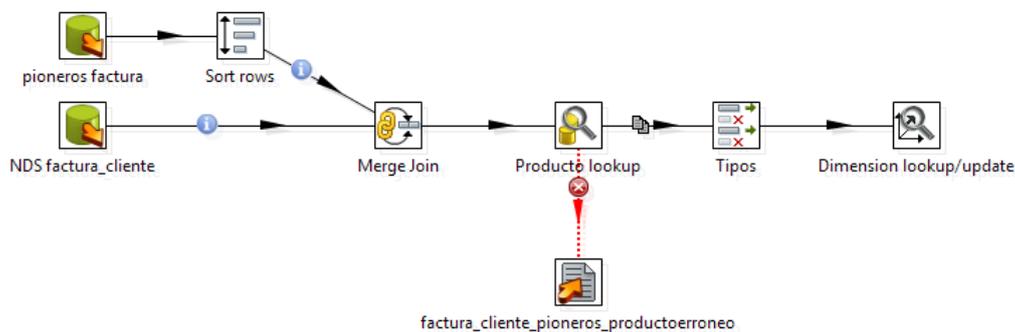


Figura 8.25: Transformación `factura_cliente_pioneros`

8.3.1.5.4 Transformación `factura_linea`

El objetivo de esta transformación (figura 8.26) es tomar aquellos datos de la tabla `factura_cli_datos_msis` de `rm_valva` que se refieran a alguna factura ya cargada en el NDS, realizando la carga en la tabla `factura_linea`. La lógica es la misma que en la transformación `factura_cliente`, con la salvedad de que hay más tipos de error, y de que en este caso se hace *lookup* del producto.

Dado que sólo puede haber una salida por el camino de tratamiento de errores (marcado con el símbolo \otimes), se emplean pasos *Dummy step* para bifurcar el flujo, enviándose todas las filas por ambos caminos.

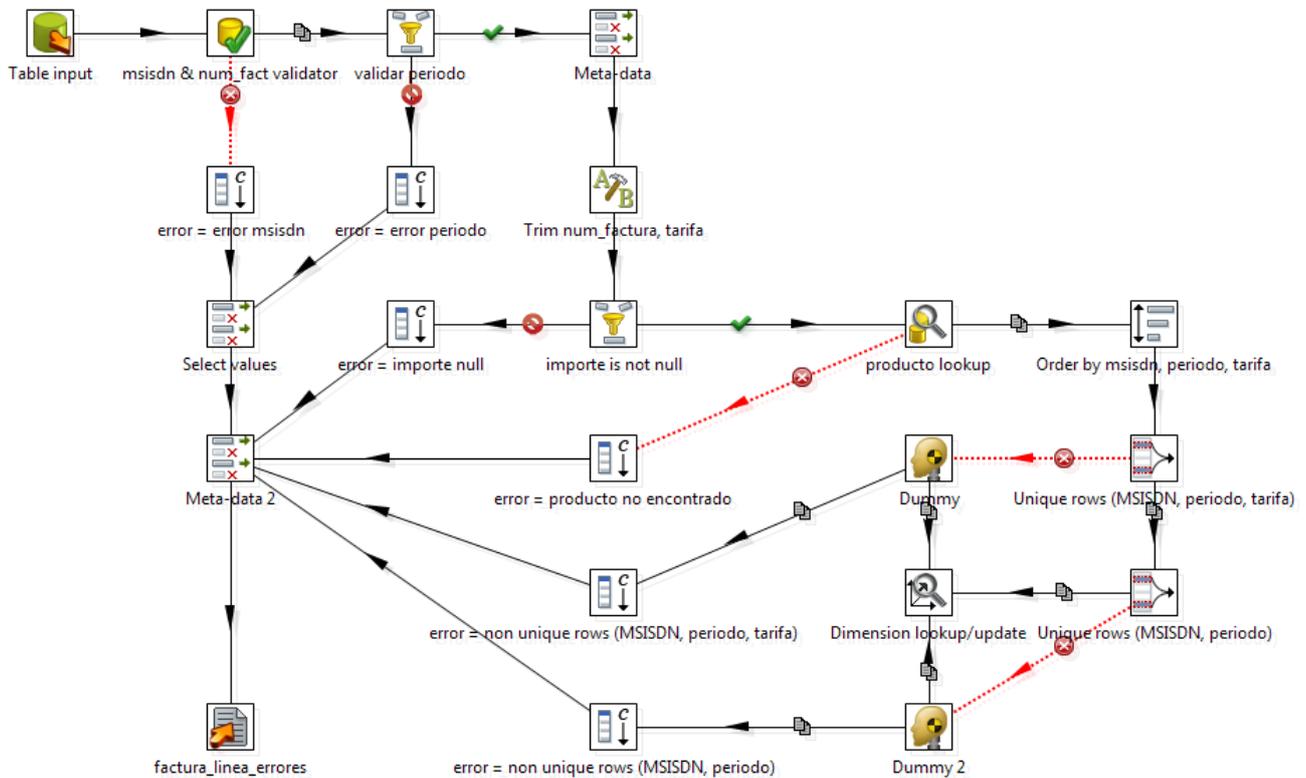


Figura 8.26: Transformación factura_linea

8.3.1.5.5 Transformación factura_linea_consumos

De forma similar a la anterior transformación, en `factura_linea_consumos` (8.27) se busca cargar un registro por línea, período y tipo de consumo en la tabla `factura_linea_consumos`. Se permite que una línea tenga registros con distintos productos ya que se podría haber producido un cambio de tarifa durante el curso del mes.

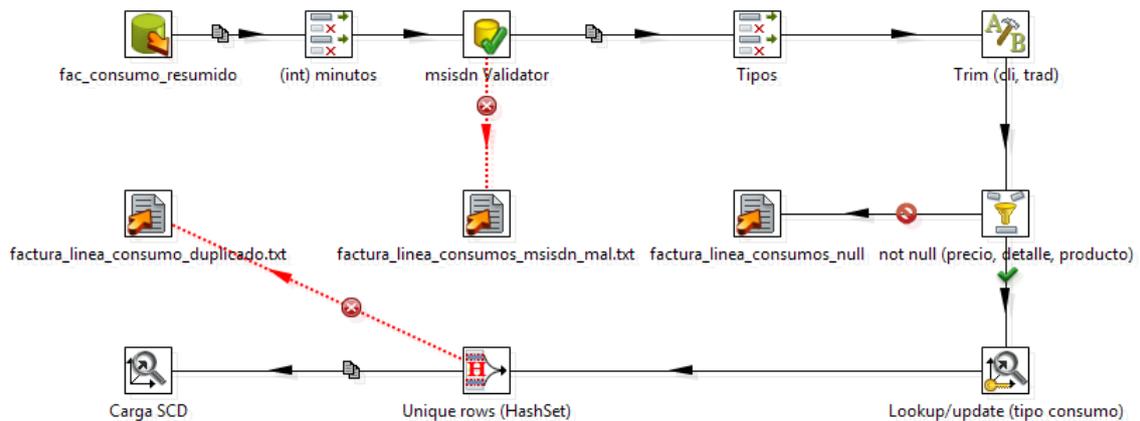


Figura 8.27: Transformación factura_linea_consumos

8.3.1.5.6 Transformación factura_promociones

Por último, se cargan las promociones aplicadas a cada factura. Únicamente se permite una fila por cliente, período y código de promoción, utilizando la siguiente sentencia para agruparlas en caso de haber más de una:

Código 8.6: Fragmento de la sentencia SELECT en el paso fact_promociones

```
1 group_concat(distinct trim(p.comentario) order by p.comentario SEPARATOR ' + ')
  ↪ as promos
2 FROM rm_walva.fact_promociones p
3 /*...*/
4 group by Id_Cliente, mes_factura, Cod_PrDatos
```

La longitud del nombre de la promoción se trunca a 255 caracteres para evitar que la transformación falle por *overflow* en el caso de que se concatenen varios nombres largos. Tras normalizar el nombre, y el canal de contratación con el que tiene que ver la promoción, se procede a la carga.

8.3.2. DDS

8.3.2.1. dim_estaticas

8.3.2.1.1 Job

En primer lugar, el *job dim_estaticas* se encarga de crear y rellenar las tablas de dimensiones *dim_provincia*, *dim_pais*, *dim_tipo_cliente*, *dim_comunidad_autonoma*, *dim_tipo_documento* y *dim_linea_estado* mediante sentencias `INSERT INTO ... SELECT FROM ...` que leen de las tablas de tipo *info_* del NDS, que habían sido rellenas manualmente. Las consultas se incluyen en el apéndice A.2.

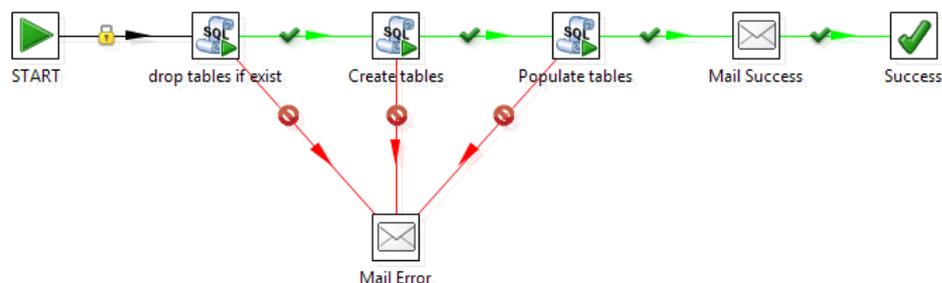


Figura 8.28: Job dim_estaticas

8.3.2.2. dim_fecha

8.3.2.2.1 Job

El objetivo de este *job* es sintetizar una dimensión *Fecha* que permita analizar los datos temporales de la forma más detallada posible. Antes de llamar a la transformación *dim_fecha*, se recuperan archivos de festivales del portal Open Data Euskadi ², acumulándose en un fichero temporal que será leído posteriormente por ésta.

²<http://opendata.euskadi.eus/catalogo-datos/>



Figura 8.29: Job dim_fecha

8.3.2.2.2 Transformación dim_fecha

Para generar desde cero una secuencia de fechas se emplean los pasos **Generate rows (fecha)**, que crea 2500 filas con un campo que contiene la fecha inicial, y **Add sequence (fecha)**, que añade una columna cuyo valor va de 0 a 2500. El paso **Formatos fecha** realiza la suma y crea los primeros campos derivados, como **year**, **month**, **day** y **day_week**. Utilizando este último, que señala el día de la semana, es posible determinar si la fecha corresponde a un fin de semana.

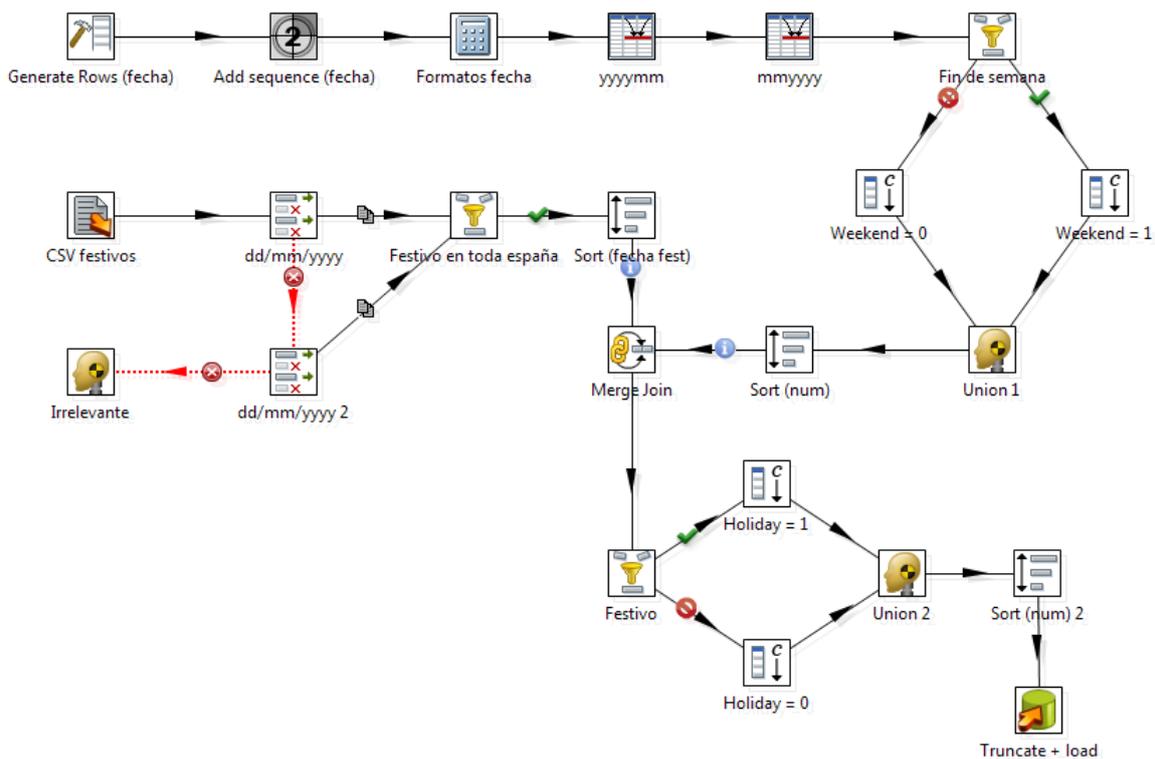


Figura 8.30: Transformación dim_fecha

Dado que los pasos de la transformación se ejecutan de forma concurrente, es necesario volver a ordenar los datos para asegurar su corrección. Mediante el paso **Merge join**, cada fecha se une a la información sobre ella que se ha obtenido del archivo de festivos. Ésta es usada a continuación para añadir el **flag holiday** que señala cuáles de las fechas son festivos nacionales. Finalmente, se trunca la tabla **dim_fecha**, y se carga de nuevo al completo.

8.3.2.3. fact_cliente

8.3.2.3.1 Job

El *job* fact_cliente tiene como objetivo realizar la carga de la tabla de hechos Cliente, llamando a la transformación homónima e informando por correo electrónico del resultado de su ejecución.

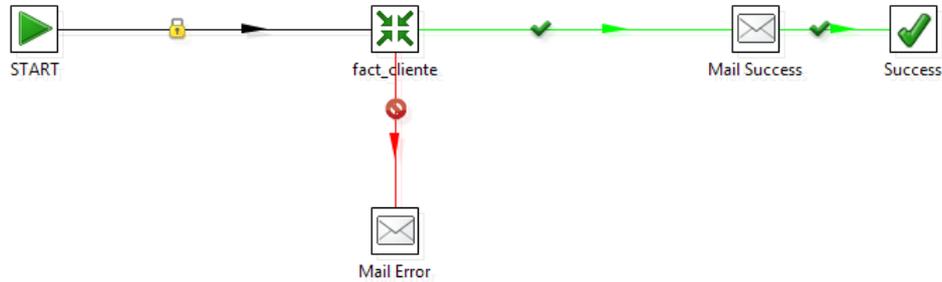


Figura 8.31: Job fact_cliente

8.3.2.3.2 Transformación fact_cliente

Para realizar esta carga, se lee información de la tabla cliente_datos del NDS a través del paso Table input cliente. En la consulta SQL que lanza se realizan algunas transformaciones, como contar cuántas líneas activas tiene en cliente_linea para cada tipo de producto. La consulta se recoge en el apéndice A.3.

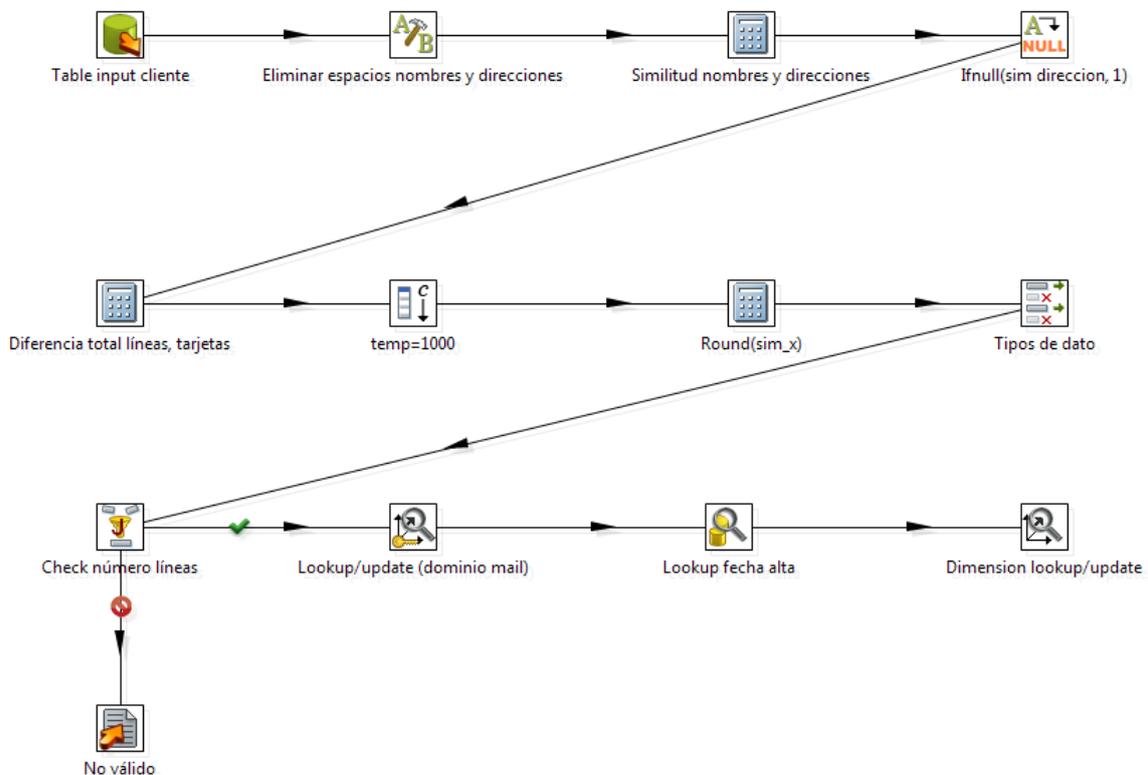


Figura 8.32: Transformación fact_cliente

En este punto comienzan a calcularse variables útiles para determinar el riesgo que supone un nuevo cliente. Entre estas medidas, está la similitud entre los nombres y direcciones de envío y facturación. En concreto, se ha calculado la similitud de Jaro-Winkler entre las dos cadenas de texto, cuyo valor puede estar entre 0 y 1. Se escogió esta variante ya que experimentalmente se vio que era la que daba los resultados más ajustados a los deseados: una similitud máxima cuando el nombre era el mismo, incluso si tenía errores tipográficos; una similitud media cuando el nombre compartía apellidos, por lo que se trataba de un familiar; y una similitud baja en el resto de los casos. En caso de contarse con una única dirección, se asume que la dirección de envío es la misma que la de facturación, por lo que la similitud tiene valor 1.

El paso **Diferencia total líneas, tarjetas** calcula el número de líneas entre las que ha contratado el cliente que tienen un origen o tipo de tarjeta SIM desconocido. Esto ayuda a encontrar datos incompletos o a revelar algunas líneas que fueron dadas de altas de forma distinta a la habitual.

A continuación, se multiplican las métricas de similitud por 1000 y se redondean, para garantizar la máxima precisión al almacenarse en base de datos. En el paso **Tipos de dato** se convierten todas las métricas a entero, detectándose descuadres en el filtro **Check número líneas**, que emplea la condición:

Código 8.7: *Java filter* Check número líneas

```
1 num_prod_mini + num_prod_peque + num_prod_peque100 + num_prod_mediana +
  ↪ num_prod_grande == num_lineas
```

Por último, se realizan *lookups* de las dimensiones **Dominio mail** y **Fecha**, y se procede con la carga.

8.3.2.4. fact_factura

8.3.2.4.1 Job

Por su parte, el *job* **fact_factura** controla el flujo de la tabla de hechos de facturación. Como en otras ocasiones, se ha utilizado una sentencia SQL para crear tablas temporales, recogida en el apéndice A.4. Estas tablas serán empleadas en la transformación **fact_factura**, y serán eliminadas cuando ésta finalice su ejecución.

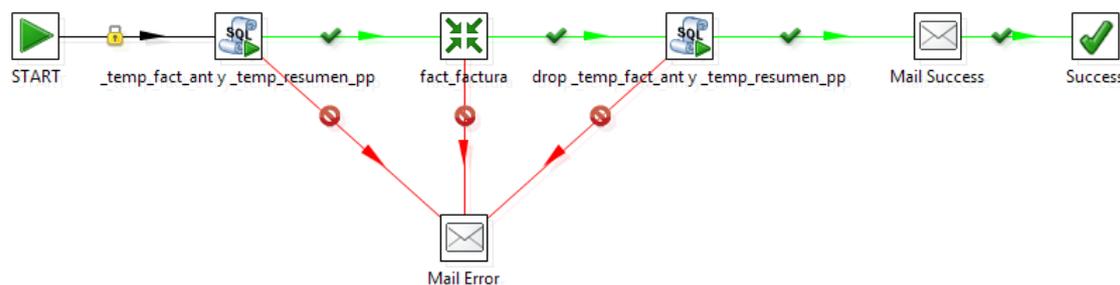


Figura 8.33: *Job* fact_factura

8.3.2.4.2 Transformación fact_factura

La mayor parte de la transformación tiene lugar en la consulta del paso *Table input*, cuyo código se detalla en el apéndice A.5. Además de calcular las variables a cargar, se calculan tres

indicadores de descuadre. En caso de que este exista, los registros se guardarán en un fichero de texto. Los correctos serán cargados directamente en la tabla, previo truncado de ésta.

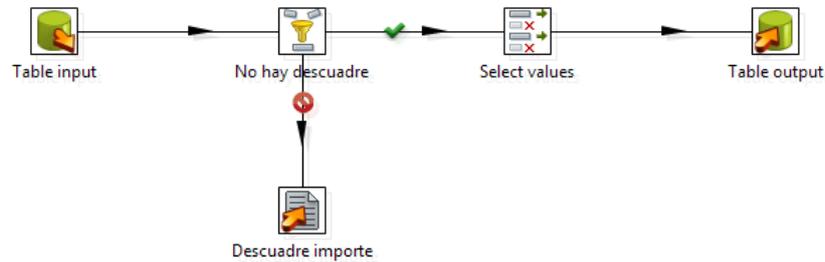


Figura 8.34: Transformación `fact_factura`

8.3.3. Reconstrucción del histórico

Dado que una de las motivaciones del proyecto de *data warehouse* era la posibilidad de mantener históricos, pero su implementación comenzó más tarde que la generación de datos, ha sido necesario reconstruir el histórico a partir de copias de seguridad.

Para llevar a cabo esta tarea, se cargó todo el repositorio de Kettle en un servidor de desarrollo. En este mismo servidor, se fueron cargando de forma sucesiva las copias de seguridad, escogiéndolas de forma que hubiera aproximadamente un mes entre dos copias.

La única modificación que fue necesario realizar sobre todas las transformaciones de este repositorio fue el cambio de la dirección del servidor desde aquel que se estaban leyendo los datos. Sin embargo, el modelo de datos de origen sufrió pequeños cambios con el tiempo, por lo que se validó cada ejecución, realizando modificaciones allá donde era necesario, hasta volver al modelo actual.

Después de cada ejecución, por otro lado, se lanzaron consultas `UPDATE` sobre los campos `date_to` y `date_from` de todas las tablas afectadas, pasando a ser iguales a la fecha de *backup*. Así, se consiguió simular un procesado a fecha pasada.

8.3.4. Exportación a Weka

La forma más sencilla de introducir datos a Pentaho Weka, el software de *data mining* a emplear, es mediante ficheros de texto. Para ello, se ha creado la transformación `weka_cliente`, que exporta todas las *features* a analizar en un formato compatible. En concreto, debe exportarse un único registro por línea, uniendo todas las dimensiones a analizar. Esto se hace a través de la consulta del apéndice A.6.

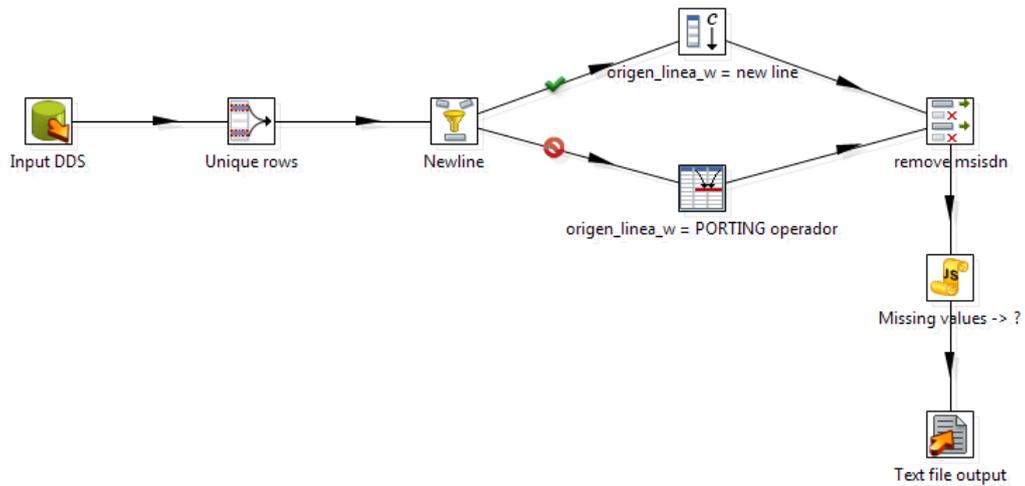


Figura 8.35: Transformación weka_cliente

La labor de la transformación es, después de asegurar que no hay duplicados, unir los campos que contienen el tipo de alta y el operador donante. Es decir, se interpreta que las nuevas altas vienen de un mismo operador donante. Después, se eliminan los datos identificativos de la línea, y se hace que los valores desconocidos pasen a ser indicados con el carácter ?. Para esto se emplea el paso *Modified Java Script Value* con el código indicado en el apéndice A.7. Por último, se exporta toda la información a un fichero CSV.

9

Diseño del sistema de *data mining*

9.1. Conceptos básicos de Pentaho Data Mining

Pentaho Data Mining, también denominado Weka [49], es un software que permite preprocesar datos, ejecutar algoritmos de *data mining* y visualizar resultados a través de una interfaz gráfica (GUI) o línea de comandos. Además, puede utilizarse como una librería externa en diversos lenguajes.

9.1.1. GUI

Como se ve en la figura 9.1, al abrir el programa se ofrece al usuario cuatro modos de trabajo:

- *Explorer*, que permite realizar tareas de pre y post procesado, ejecutar algoritmos y visualizar resultados de la forma más sencilla, configurándose cada proceso de forma manual.
- *Experimenter*, a través del cuál es posible comparar diversos métodos de aprendizaje de una forma más automática.
- *Knowledge flow*, que permite realizar tareas similares a las de *Explorer*, con la diferencia de que se estructuran como flujos de trabajo creados en una interfaz *drag and drop*.
- *Simple CLI*, que proporciona una interfaz de línea de comandos a través de la cuál trabajar en sistemas operativos que no ofrecen esta posibilidad.

Además de acceder a los componentes principales, es posible descargar e instalar paquetes adicionales desde *Tools > Package manager*. Esto incluye métodos de procesado y visualización y algoritmos, muchos de ellos creados por la comunidad.

De acuerdo con el manual, el procedimiento recomendado es realizar los ajustes a través del *Explorer*, debido a su simplicidad, y ejecutar a través de línea de comandos, ya que el consumo de memoria es menor y, por tanto, el rendimiento es mayor.

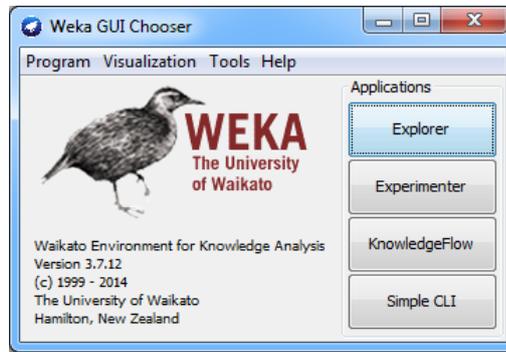


Figura 9.1: Pantalla inicial de WEKA

Tras cargar el archivo CSV exportado a través de `weka_cliente`, el aspecto del *Explorer* es el de la figura 9.2. El módulo mostrado es el de preprocesado, que proporciona herramientas de análisis exploratorio y posibilita la aplicación de filtros a los datos de entrada, visualizándose de inmediato su efecto.

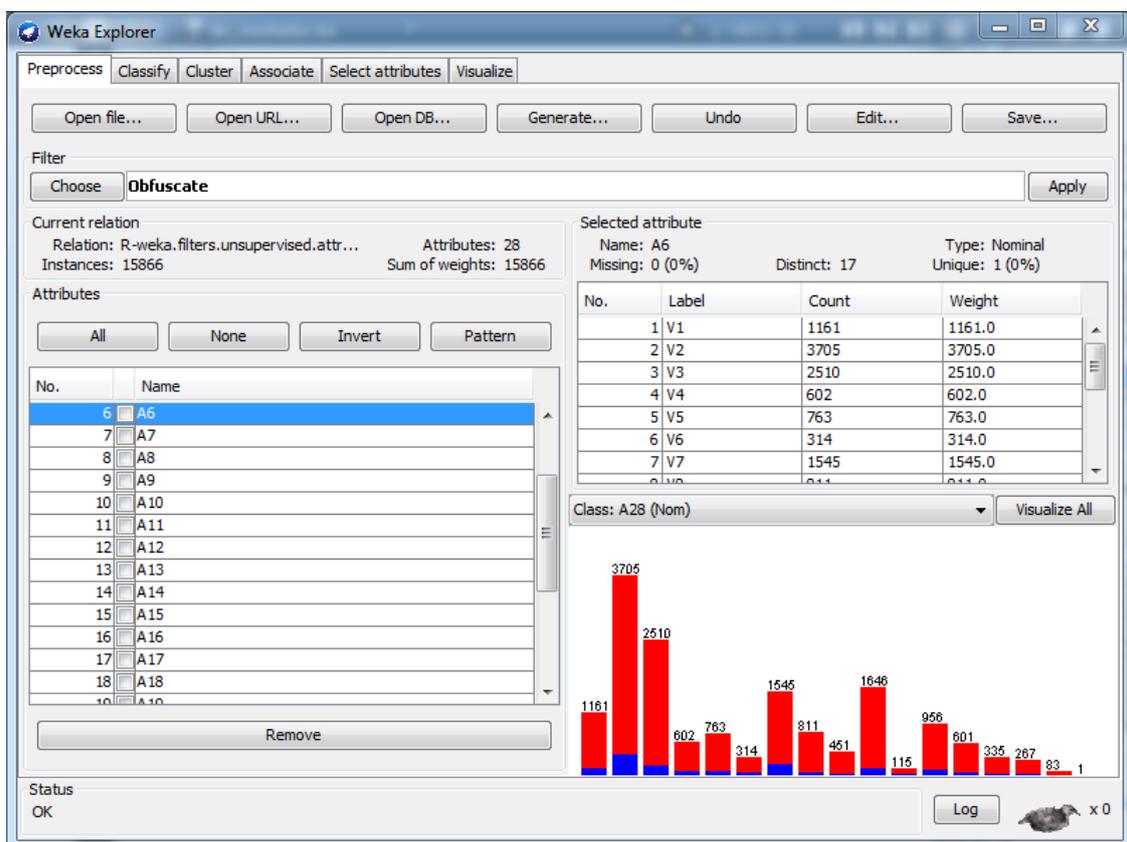


Figura 9.2: Módulo de preprocesado del *Explorer*

Los filtros disponibles se clasifican en supervisados y no supervisados, según tengan o no en cuenta la clase. Por otro lado, algunos se aplican sobre atributos individuales, y otros sobre la instancia, es decir, todo el *dataset*.

Al seleccionar uno de los atributos, la sección *selected attribute* muestra un conteo de los valores distintos asociados a éste. Cuando el atributo es numérico, se muestran el máximo, el mínimo, la media y la desviación típica. Además, detalla la cantidad de valores desconocidos. En la esquina inferior derecha se muestra un histograma, diferenciando las clases positiva y negativa con el color.

Los módulos de clasificación, *clustering* y asociación permiten la aplicación de algoritmos. Haciendo clic sobre el botón *Choose*, se despliega una lista con todos los métodos disponibles para este *dataset*.

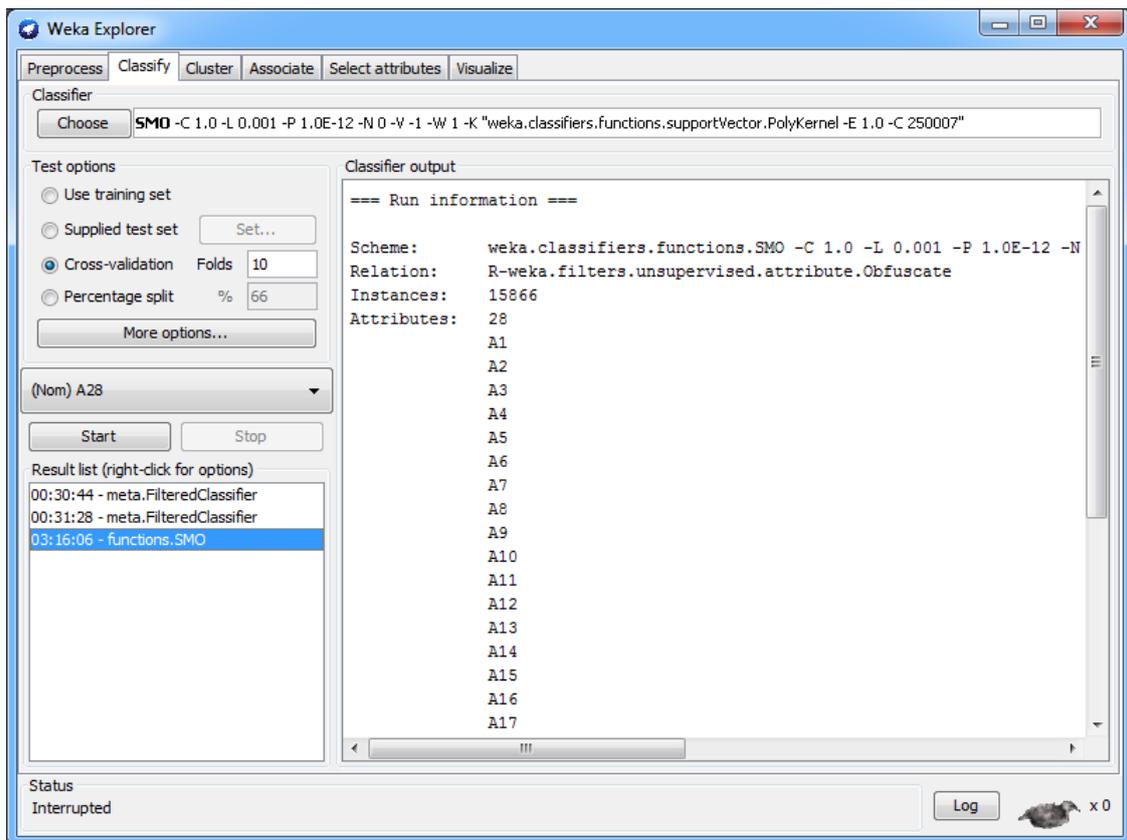


Figura 9.3: Módulo de clasificación del *Explorer*

Haciendo clic sobre la barra que contiene los parámetros de ejecución aparecen ventanas que permiten configurar los argumentos de forma más sencilla. Este comando puede copiarse y utilizarse para ejecutar con los mismos parámetros desde un programa Java o desde línea de comandos.

En la parte izquierda, por otro lado, pueden escogerse las opciones del conjunto de test, como los *fold*s de *cross validation*, o la fracción del *training set* que debe usarse si se escoge muestrear a partir de él.

Por último, el módulo *Select attributes* puede emplearse para aplicar técnicas de reducción de dimensionalidad, mientras que *Visualize* permite representar gráficamente dos variables en un plano, diferenciando sus clases mediante el color en el que se muestran.

9.1.2. Weka como librería externa

Además del uso mediante interfaz gráfica, Weka puede ser importado a código Java. En primer lugar, en un IDE como Eclipse, debe añadirse al *classpath*. Para esto, debe seleccionarse el archivo *weka.jar* en la pestaña *Classpath*, que puede encontrarse en el menú *Run > Run configurations...*, como se puede ver en la figura 9.4.

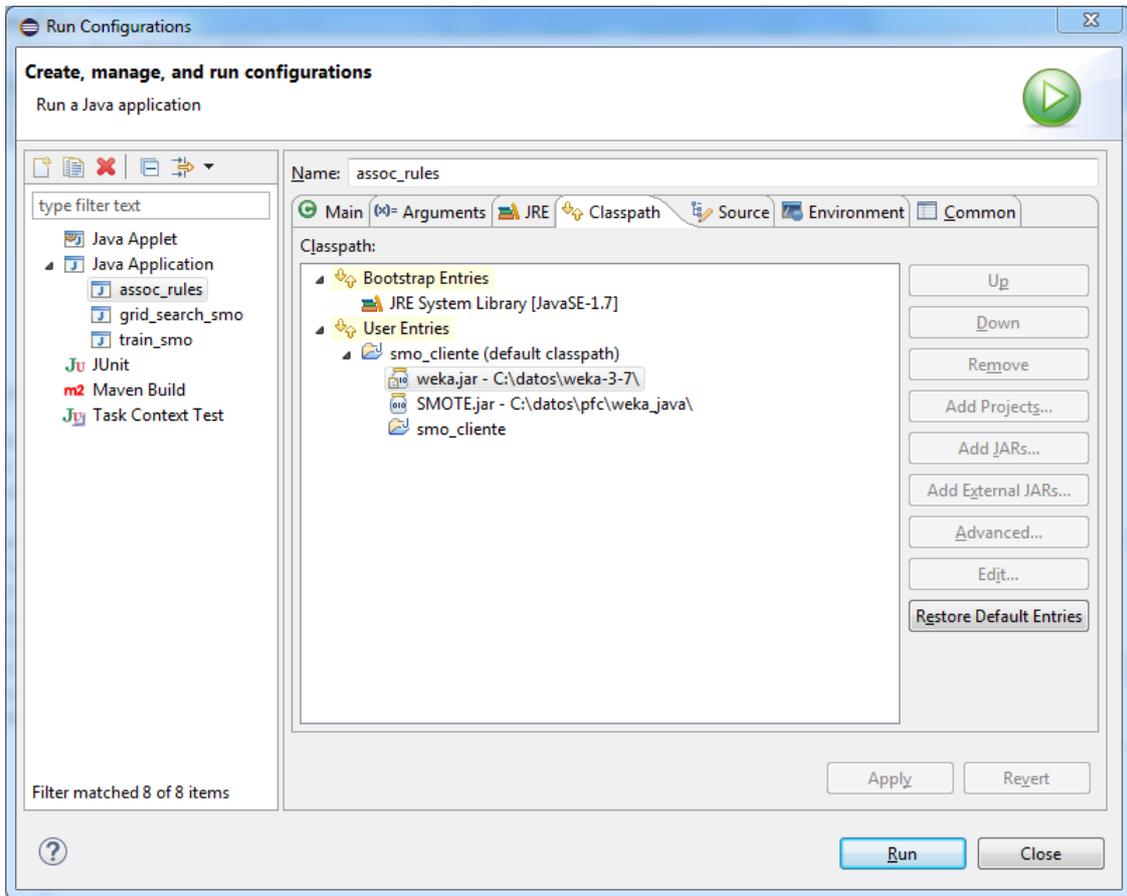


Figura 9.4: *Run configurations* en Eclipse IDE

Una vez la configuración ha sido realizada, los componentes de Weka aparecen en el *Package explorer*, como muestra la figura 9.5. Las clases necesarias pueden ser importadas mediante una sentencia `import`.

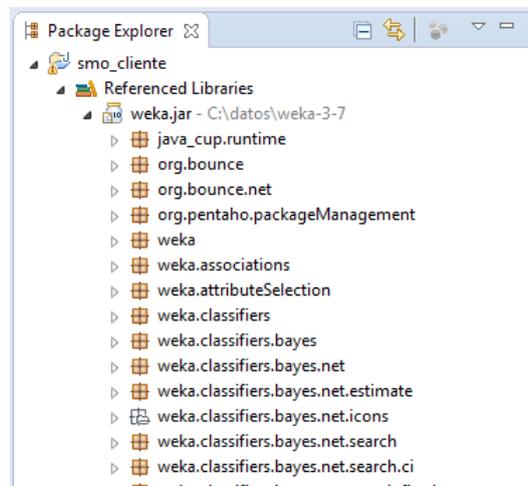


Figura 9.5: *Package explorer* en Eclipse IDE

9.2. Sistema implementado

El sistema desarrollado tiene como objetivo evaluar el rendimiento del clasificador sobre múltiples divisiones aleatorias del conjunto de datos. El esquema se muestra en la figura 9.6. Los bloques recogidos dentro del rectángulo punteado se ejecutan un total de 10 veces, haciéndose finalmente la media de las medidas (*g-means*, AuC...) obtenidas. Con este fin se ha creado la clase `evaluate_smo`, cuyo código se detalla en el apéndice A.8.

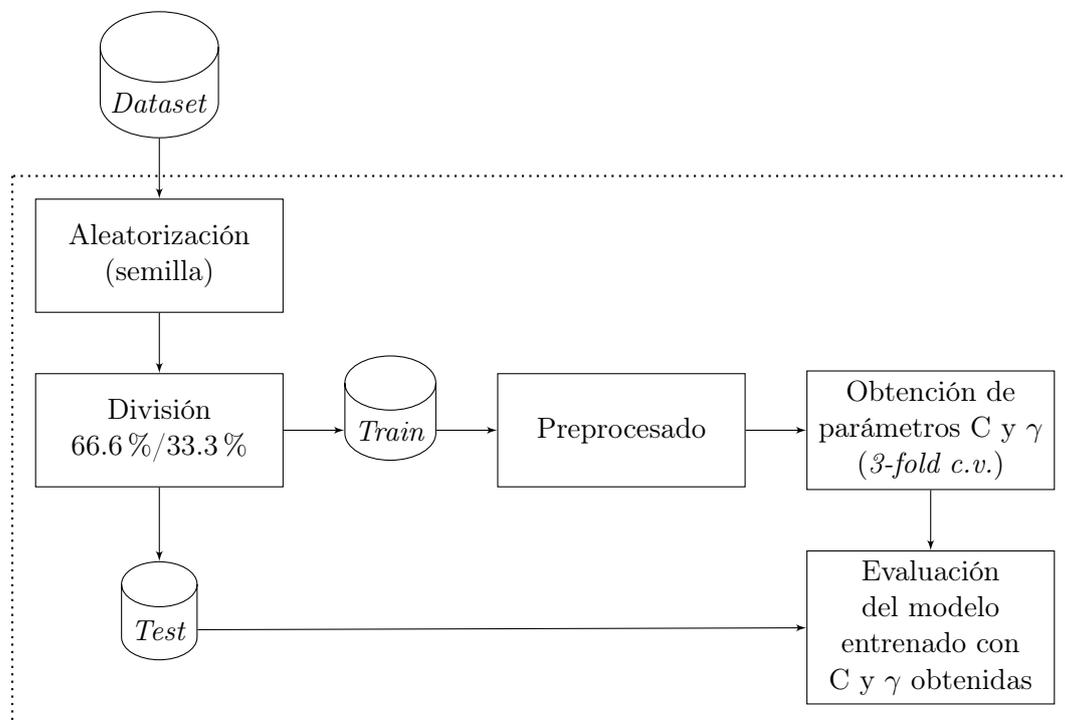


Figura 9.6: Esquema del sistema a través del cuál se ha evaluado el modelo

9.2.1. Lectura del conjunto de datos, aleatorización y división

Para capturar el conjunto de datos se emplea la clase `load_csv` de Weka, que toma los datos de un fichero CSV y los guarda en un objeto de tipo `Instances`. El nombre del archivo del que leer se obtiene a través de un argumento.

A la hora de evaluar un modelo es necesario asegurarse de que éste no ha sido expuesto a datos de entrenamiento. Por ello, es necesario dividir el conjunto de datos antes de optimizar los parámetros del SVM. Para esto, se sigue el procedimiento recomendado en el wiki de Weka [50]. Se aleatorizan los datos pre-procesados, y se guardan dos nuevos conjuntos en distintos objetos, albergando el conjunto de entrenamiento el 66.66% de los registros, y el 33.33% restante el de test.

9.2.2. Pre-procesado

De cara al pre-procesado se han creado tres métodos: `pre_process_train`, `pre_process_test` y `pre_process_common`, aplicándose el primero antes del muestreo que divide el *dataset* entre conjuntos de entrenamiento y test, mientras que los dos posteriores se emplean sobre estos conjuntos por separado.

Todos los datos de entrada se procesan mediante el método `pre_process_common`. En primer lugar, se escoge el atributo que contiene la etiqueta de la clase. Después, se eliminan los atributos que contienen una discretización manual de las similitudes entre el titular y la dirección de facturación y envío, ya que inmediatamente después se utiliza el método automático de Fayyad e Irani, explicado en la sección 4.2.4. De forma teórica, la aplicación de este algoritmo sobre datos de entrenamiento y test supone tener en cuenta datos que no deberían conocerse, pero se asume por simplicidad que las fronteras de decisión van a ser las mismas, ante una distribución similar en ambos conjuntos.

Tras la división en conjuntos de entrenamiento y test puede pasarse a finalizar el pre-procesado. Por un lado, se ejecutan *SMOTE*, *Class Balancer* y *Replace missing values* sobre el conjunto de entrenamiento. Tal y como se explicó en la sección 4.2.6, *SMOTE* es un algoritmo de sobremuestreo. Por otro lado, el filtro *Class Balancer* aplica pesos distintos a las clases, haciendo que el algoritmo penalice más los errores de la clase con menor representación. Este es un método para tratar con *datasets* no balanceados equivalente a usar un clasificador sensible a costes [51].

Por último, se ejecuta un método para rellenar valores incompletos. En la primera prueba se emplea *Replace Missing Values*, ya incluido en Weka, que sigue el algoritmo MC, analizado en la sección 4.2.3.

La segunda prueba consiste en la utilización de reglas asociativas. Para esto, se ha desarrollado el método `getAttrMissing`, que examina los atributos uno a uno, almacenando el número de valores desconocidos y devolviendo los atributos ordenados de forma descendente en función de éste número. A continuación, se llama al método `getInstMV`, que devuelve los registros con valores desconocidos para una lista de atributos dada.

Para cada atributo con valores desconocidos del *dataset*, se buscan reglas asociativas que predigan su valor a partir del resto de atributos. Esto tiene lugar en el método `fillMV`, que a continuación tratará de rellenar las instancias recibidas de `getInstMV` con las reglas obtenidas. Para ello, se marca el valor desconocido como clase, y se utiliza el modo *Class Association Rules*, de forma que se busquen reglas que predigan esa clase. La única métrica soportada por este modo es la confianza, cuyo valor mínimo puede definirse en los argumentos.

Por último, se guarda el fichero con los valores rellenados como CSV. Este archivo puede ser leído por la clase `evaluate_smo`, indicándose su ruta como argumento.

Cabe destacar que sobre el conjunto de test únicamente se ejecuta el proceso de rellenado de valores incompletos, ya que aplicar los otros dos filtros no tiene sentido.

9.2.3. Optimización de parámetros del clasificador

Como se comentó en la sección 4.5.2, el procedimiento recomendado para encontrar los parámetros C y γ del SVM es utilizar un método de búsqueda en malla (*grid search*). Para ello puede emplearse el paquete `weka.classifiers.meta.multisearch`, que permite realizar una búsqueda del mejor conjunto de valores en un número arbitrario de parámetros, evaluando los resultados de la clasificación respecto a una métrica a escoger. Este procedimiento debe lanzarse sobre el conjunto de entrenamiento, de forma que los parámetros no se optimicen para los datos de test.

Ya que el código de este metaclasificador puede encontrarse libremente en Github¹, ha sido posible añadir el código necesario para calcular *g-means*, definiendo esa medida como la métrica con respecto a la cuál evaluar el desempeño de los parámetros. Un ejemplo de comando para ejecutarlo es el que aparece en el código 9.1:

¹<https://github.com/fracpete/multisearch-weka-package>

Código 9.1: Opciones del clasificador MultiSearch

```
1 options = weka.core.Utils.splitOptions("-E GM -search \"weka.core.setupgenerator
  ↪ .MathParameter -property classifier.c -min -1.0 -max 3.0 -step 2.0 -base
  ↪ 2.0 -expression pow(BASE,I)\" -search \"weka.core.setupgenerator.
  ↪ MathParameter -property classifier.kernel.gamma -min -7.0 -max -3.0 -step
  ↪ 2.0 -base 2.0 -expression pow(BASE,I)\" -sample-size 100.0 -log-file C
  ↪ :\\datos\\test_new_\"+seed +\".txt -initial-folds 3 -subsequent-folds 0 -
  ↪ num-slots 4 -S 1 -W weka.classifiers.functions.SMO -- -C 1.0 -L 0.001 -P
  ↪ 1.0E-12 -N 0 -V -1 -W 1 -K \"weka.classifiers.functions.supportVector.
  ↪ RBFKernel -G 0.01 -C 250007\");
```

Como se puede apreciar, el primer paso es definir el método de evaluación, escogiéndose *g-means* con el argumentos `-E GM`. Posteriormente, se define el mallado. En concreto, con la opción `-search` se hace que C tome valores entre 2^{-1} y 2^3 y que γ valga entre 2^{-7} y 2^{-3} , ambos variando con pasos de 2^2 . A continuación, `-log-file` define el lugar donde se va a guardar el *log* de ejecución, mientras que el número de *folds* del método de *cross validation* se configura en los argumentos `-initial-folds` y `-subsequent-folds`. En particular, en primer lugar se lanzarán *3-fold cross validations* sobre todo el mallado. Mediante `-num-slots` se escoge la cantidad de hilos que van a lanzarse de forma concurrente.

Por último, debe escogerse el método a lanzar. Para ello se emplea el argumento `-W`, donde se escoge el paquete `SMO`. De encontrarse que el punto que da mejor resultado está en el borde del mallado, es posible ampliarlo sin volver a ejecutarlo por completo, escogiendo la misma semilla y modificando las opciones de `MultiSearch`.

9.2.4. Evaluación

`MultiSearch` devuelve los valores de la mejor pareja de parámetros encontrada, y un objeto de tipo `Classifier` que puede ser entrenado con el método `buildClassifier`. Tras hacerlo, el modelo se guarda para realizar pruebas posteriores de ser necesario. Estos modelos pueden cargarse y evaluarse de nuevo tanto en la GUI como a por línea de comandos.

A continuación, el modelo se evalúa sobre los datos de test. Los objetos `Evaluation` contienen la matriz de confusión, que incluye verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. A partir de estos datos pueden calcularse las métricas que resulten de mayor interés. En este caso se ha optado por *g-means*, AuC, precisión positiva y precisión negativa, utilizando las expresiones de la sección 4.5.2.

Todos los resultados se guardan en un log, junto a la semilla que los ha generado, para su posterior análisis.

10

Resultados

En este capítulo se recogen los resultados obtenidos al evaluar el modelo predictivo construido.

10.1. Configuración de los experimentos

El proceso para probar los distintos algoritmos de aprendizaje ha consistido en la realización de 10 divisiones aleatorias del conjunto de datos en entrenamiento y test. El tamaño de las particiones ha sido del 66.6% y 33.3%, respectivamente.

Para cada una de estas particiones, como se explicó en la sección 9.2, el primer paso del método de evaluación ha consistido en ejecutar una búsqueda en malla de los parámetros C y γ . Durante este proceso, se ha definido un mallado de parámetros, siendo cada punto una pareja de C y γ que van a ser utilizados para entrenar el modelo. Tras la ejecución, se evalúa el rendimiento.

El objetivo de este procedimiento es encontrar los parámetros que mejor resultado dan en cuanto a *g-means*, mediante validación cruzada con 3 iteraciones dentro del conjunto de entrenamiento.

Una vez encontrados los parámetros óptimos para la SVM, se entrena este modelo usando dichos parámetros y todos los datos de entrenamiento. Además se ha entrenado el modelo de bosques aleatorios con 150 árboles. Adicionalmente, cada modelo (SVM y BA) ha sido probado usando imputación de valores desconocidos mediante MC (*Most Common*) y reglas asociativas.

Por último se usa el conjunto de test para validar los modelos obtenidos calculando la precisión global (Acc), la precisión en las clases positiva (Acc+) y negativa (Acc-), gmeans y AuC.

10.2. Modelo basado en SVM

La tabla 10.1 recoge la media de esta medida para cada punto del mallado. Como se puede ver, el mejor punto entre los calculados es $(C, \gamma) = (2^3, 2^{-5})$.

$\log_2(C)$	$\log_2(\gamma)$			
	-7	-5	-3	-1
-1	0.7703	0.7956	0.8285	0.7354
1	0.7885	0.8232	0.8340	0.7120
3	0.8098	0.8348	0.8135	0.7093
5	0.8264	0.8298	0.8067	0.7116

Tabla 10.1: Resultados de `MultiSearch` (*3-fold cross validation*)

La tabla 10.2 recoge los valores de precisión global y para cada clase, *g-means* y área bajo la curva ROC obtenidos en cada una de las diez ejecuciones así como el valor medio y la desviación típica.

Como se puede ver en los resultados, las ejecuciones que mejor *g-means* producen son aquellas en las que la precisión sobre la clase negativa aumenta, incluso cuando la precisión sobre la clase negativa disminuye. También se aprecia que, como se esperaba, en un conjunto de datos no balanceado como este ni la AuC ni la precisión global son buenas medidas para determinar la calidad del clasificador. En las 10 ejecuciones realizadas, se ha obtenido una *g-means* media del 69.59 %, y una AuC media de 0.8318.

<i>Seed</i>	Acc	Acc+	Acc-	<i>g-means</i>	AuC
8	0.8485	0.8721	0.6320	0.7424	0.8450
20	0.8875	0.9314	0.4836	0.6712	0.8298
23	0.8905	0.9318	0.4990	0.6819	0.8390
32	0.8589	0.8848	0.6187	0.7399	0.8351
49	0.8425	0.8649	0.6190	0.7317	0.8235
58	0.8601	0.8891	0.5927	0.7259	0.8355
60	0.8915	0.9395	0.4695	0.6642	0.8290
63	0.8818	0.9263	0.4811	0.6675	0.8286
70	0.8913	0.9364	0.4534	0.6516	0.8246
97	0.8960	0.9396	0.4962	0.6828	0.8278
Media	0.8748	0.9116	0.5345	0.6959	0.8318
Desv. típica	0.0202	0.0301	0.0716	0.0350	0.0068

Tabla 10.2: Evaluación del modelo con los parámetros encontrados con `MultiSearch` frente al conjunto de test.

10.3. Modelo basado en SVM con imputación de valores desconocidos mediante reglas asociativas

En segundo lugar se ha comprobado que el método descrito en 9.2.2 para rellenar los valores desconocidos mediante reglas asociativas es capaz de mejorar la capacidad predictiva del modelo. Para ello, se ha utilizado el código incluido en el apéndice A.9 para generar un nuevo conjunto de datos a partir del anterior, incluyendo predicciones de los valores que faltan. Este fichero se ha utilizado para alimentar el mismo código que el que se ha empleado para evaluar el modelo en la sección 10.2.

En primer lugar, se ha buscado la pareja de parámetros C y γ con la que se obtiene mejor *g-means*. Como se puede ver en la tabla 10.3, de nuevo se ha obtenido $(C, \gamma) = (2^3, 2^{-5})$, con una *g-means* ligeramente mayor que en la sección anterior.

		$\log_2(\gamma)$		
		-7	-5	-3
$\log_2(C)$	-1	0.7684	0.7953	0.8340
	1	0.7881	0.8246	0.8354
	3	0.8099	0.8370	0.8146
	5	0.8258	0.8288	0.8037

Tabla 10.3: Resultados de `MultiSearch` (*3-fold cross validation*)

A continuación, se ha evaluado el mejor modelo de cada ejecución contra el conjunto de test. Tal y como se muestra en la tabla 10.4, se ha obtenido una *g-means* media de 71.64 %, suponiendo una mejora del 2 % sobre el método MC de rellenado de valores incompletos. También se aprecia mejoría en la AuC, pasando de 0.8318 de MC a 0.8338 al utilizar reglas asociativas.

<i>Seed</i>	Acc	Acc+	Acc-	<i>g-means</i>	AuC
8	0.8468	0.8719	0.6166	0.7332	0.8455
20	0.8557	0.8830	0.6050	0.7309	0.8404
23	0.8572	0.8806	0.6356	0.7482	0.8400
32	0.8593	0.8869	0.6031	0.7314	0.8332
49	0.8903	0.9315	0.4803	0.6689	0.8157
58	0.8534	0.8826	0.5849	0.7185	0.8297
60	0.8635	0.8928	0.6063	0.7357	0.8388
63	0.8553	0.8845	0.5928	0.7241	0.8334
70	0.8913	0.9370	0.4474	0.6474	0.8239
97	0.8608	0.8901	0.5923	0.7261	0.8374
Media	0.8634	0.8941	0.5764	0.7164	0.8338
Desv. típica	0.0151	0.0219	0.0615	0.0321	0.0088

Tabla 10.4: Evaluación del modelo con los parámetros encontrados con `MultiSearch` frente al conjunto de test.

El único parámetro a optimizar en el método de reglas asociativas es la confianza mínima de las reglas. La utilización de una confianza mínima baja (0.5) hace que el algoritmo rellenara los valores con la media y moda para cada atributo, con lo que los resultados obtenidos son los mismos que para MC. Por ello, no se recogen en esta sección. Los resultados aquí reunidos se han conseguido con una confianza mínima de 0.9.

10.4. Modelo basado en bosques aleatorios

A modo comparativo, se ha evaluado un modelo basado en bosques aleatorios (*Random forests*). El único parámetro en este caso es el número de bosques, que se ha fijado en 150. De nuevo, se ha ejecutado sobre 10 divisiones aleatorias del conjunto de datos, obteniendo los resultados de la tabla 10.5 al rellenar los valores incompletos mediante el método MC.

Como se puede ver, se ha obtenido una *g-means* de 70.27 % y una AuC de 0.8329. Los resultados son mejores que los de SVM con método MC para rellenado de valores incompletos, pero peores que este clasificador al emplear reglas asociativas para completar los datos.

<i>Seed</i>	Acc	Acc+	Acc-	<i>g-means</i>	AuC
8	0.8837	0.9193	0.5568	0.7155	0.8382
20	0.8820	0.9197	0.5356	0.7019	0.8254
23	0.8916	0.9264	0.5624	0.7218	0.8403
32	0.8869	0.9221	0.5603	0.7188	0.8361
49	0.8826	0.9188	0.5217	0.6924	0.8323
58	0.8812	0.9189	0.5347	0.7010	0.8277
60	0.8829	0.9258	0.5065	0.6848	0.8321
63	0.8775	0.9149	0.5398	0.7027	0.8366
70	0.8826	0.9195	0.5243	0.6943	0.8250
97	0.8835	0.9230	0.5212	0.6936	0.8356
Media	0.8835	0.9208	0.5363	0.7027	0.8329
Desv. típica	0.0037	0.0035	0.0188	0.0123	0.0054

Tabla 10.5: Evaluación del modelo de bosques aleatorios con rellenado de valores desconocidos por MC.

Si, por el contrario, se emplean reglas asociativas para rellenar los valores, se obtienen los resultados de la tabla 10.6. Si bien se aprecia una pequeña mejoría en *g-means* frente a MC, ésta es bastante menor que la que se producía con el clasificador SVM. En este caso, se ha pasado de 70.27% a 70.37%. Sin embargo, la precisión global y la AuC han disminuido levemente.

La curva ROC (*Receiver Operating Characteristic*) del modelo de bosques aleatorios con imputación por MC se muestra en la figura 10.1. En ella se representan la tasa de verdaderos positivos (eje vertical) y falsos positivos (eje horizontal) en función del umbral de decisión del clasificador, cuyo valor viene determinado por el color de la curva. Por defecto, Weka utiliza un umbral de 0.5. Este punto se muestra en rojo en la gráfica.

Es posible escoger un umbral distinto para hacer que las predicciones sean más adecuadas a las políticas de la compañía. La elección de un valor mayor supone un aumento en el número de muestras para las que se predice clase negativa, mientras que un valor menor hace que más muestras se predigan como clase positiva, con lo que habrá mayor tasa de falsos positivos. Es decir, se tomarán posiciones más arriesgadas cuando el umbral sea más alto, mientras que las predicciones serán más conservadoras conforme el umbral disminuya, ya que el modelo tenderá más a clasificar como impagador.

<i>Seed</i>	Acc	Acc+	Acc-	<i>g-means</i>	AuC
8	0.8820	0.9180	0.5511	0.7113	0.8399
20	0.8765	0.9151	0.5222	0.6912	0.8203
23	0.8888	0.9224	0.5703	0.7253	0.8372
32	0.8837	0.9177	0.5681	0.7220	0.8336
49	0.8833	0.9197	0.5217	0.6927	0.8277
58	0.8809	0.9184	0.5347	0.7008	0.8271
60	0.8818	0.9237	0.5139	0.6890	0.8351
63	0.8812	0.9179	0.5511	0.7112	0.8369
70	0.8799	0.9159	0.5304	0.6970	0.8215
97	0.8788	0.9169	0.5288	0.6964	0.8295
Media	0.8817	0.9186	0.5392	0.7037	0.8309
Desv. típica	0.0033	0.0027	0.0198	0.0130	0.0067

Tabla 10.6: Evaluación del modelo de bosques aleatorios con rellenado de valores desconocidos mediante reglas asociativas.

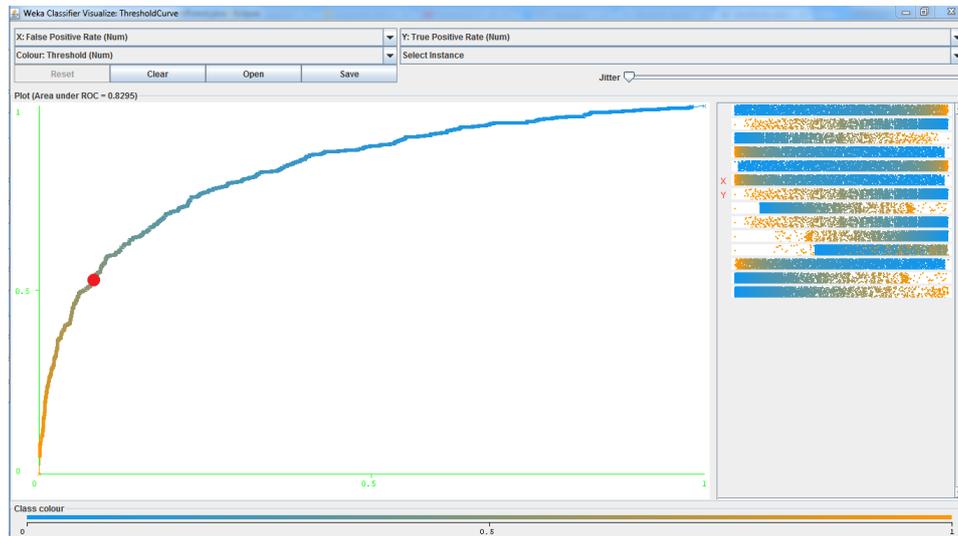


Figura 10.1: Curva ROC del modelo de bosques aleatorios. El eje horizontal es la tasa de falsos positivos, y el eje vertical es la tasa de verdaderos positivos.

10.4.1. Discusión de los resultados

En conclusión, el sistema que mejores resultados presenta en cuanto a *g-means* es el de SVM con relleno de valores incompletos mediante reglas asociativas. Una ventaja de este método es que, al ser no supervisado, permite aprovechar una mayor cantidad de información intrínseca a los conjuntos de entrenamiento y test, ya que el conjunto de datos completo se emplea para rellenar la información desconocida. Por otro lado, los bosques aleatorios obtienen mejor resultado en precisión global y precisión sobre la clase positiva y tienen la ventaja de ser muy rápidos a la hora de entrenar ya que no es necesario estimar ningún parámetro.

11

Conclusiones y trabajo futuro

En este capítulo se resume el desarrollo del proyecto, el aprendizaje a lo largo de sus fases y los retos a resolver en el futuro.

La mayor parte del tiempo invertido en este proyecto se ha dedicado a la preparación de los datos. El hecho de que existan múltiples fuentes de datos, siendo cada una de ellas fruto de distintos requisitos técnicos y funcionales hace que su consolidación sea un reto. Es inevitable que aparezcan descuadres en la información, siendo su resolución especialmente compleja en los casos en que no se tiene certeza de la información correcta.

Afortunadamente, en este caso se cuenta con una fuente de información prioritaria como la base de datos del MVNE. A lo largo del desarrollo, se ha visto que la mayoría de incongruencias se deben a datos que quedan desactualizados, ya sea porque dejan de utilizarse, porque se ha modificado la forma de calcularlos o porque fueron generados por procedimientos automáticos que ya han sido decomisados. También es común que se realicen cambios manuales como respuesta a peticiones *ad-hoc*.

Para evitar estos problemas, es recomendable emplear un modelo de datos claro, que pueda integrar con facilidad nueva información conforme el negocio evolucione. Además, es interesante considerar la realización de auditorías que evalúen periódicamente la calidad de los datos, detectando descuadres y eliminando aquellas tablas que ya no sean empleadas. De esta forma, la probabilidad de error en los registros se reduce considerablemente.

Durante esta fase de preparación de datos, se ha desplegado un *data warehouse* flexible y fiable. La información proveniente de los diversos sistemas operacionales ha sido consolidada en un *Normalized Data Store*, que sirve de etapa intermedia para crear tantos *Dimensional Data Stores* como sea necesario. En concreto, se ha construido un *datamart* que permite analizar los impagos en función de varias dimensiones.

Para esto se ha empleado Pentaho Kettle. Las cargas se han programado en horario nocturno, llevando su ejecución aproximadamente una hora. Este tiempo de carga es extremadamente satisfactorio, ya que deja margen de sobra para incorporar procesos nocturnos adicionales, como copias de seguridad.

En paralelo al desarrollo de esta parte del proyecto, se ha configurado un servidor Pentaho BI Server que permite configurar y visualizar informes a partir de la información del *Dimensional Data Store*. Se ha empleado tanto para validar la calidad de la información contenida en este,

como la eficiencia del acceso a ésta, ya que las consultas pueden llevar varios minutos cuando el diseño de la base de datos no es el apropiado para una aplicación de BI.

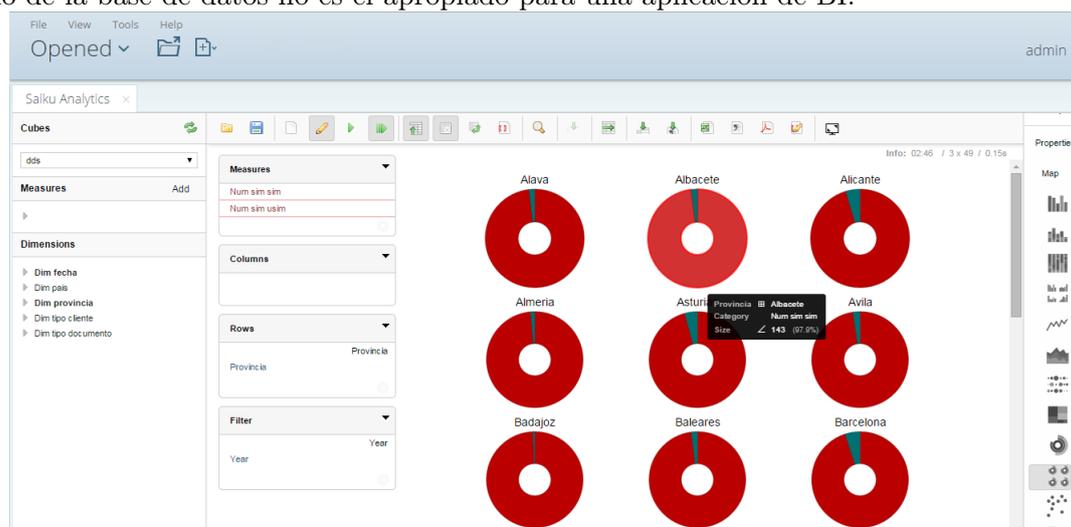


Figura 11.1: Pantallazo de Saiku Analytics, un *plug-in* de Pentaho BI Server conectado al DDS implementado.

Como puede verse en la figura 11.1, la aplicación ha quedado lista para utilizarse para realizar distintos análisis. Al utilizar el método de Kimball, se ha garantizado que los reportes cargan suficientemente rápido –entre 1 y 10 segundos, según la información que contengan.

La predicción se ha llevado a cabo utilizando 27 atributos conocidos en el momento del alta de 15000 clientes. Como se comentó en la sección 9.2.2, el pre-procesado que se ha realizado incluye discretización, SMOTE, balanceo de clases y rellenado de datos desconocidos mediante MC.

Tanto el balanceo de clases como SMOTE han resultado críticos para obtener un clasificador de utilidad. Al suprimirlos durante la realización de pruebas, se encontró que se obtenía una precisión exactamente igual a la proporción de muestras negativas.

Los valores incompletos resultan problemáticos en buen número de *datasets*: es bastante común que algunos valores falten, ya sea porque el usuario ha decidido no proporcionarlos, no se hayan medido o sean consecuencia de algún error o descuadre en el sistema. Por ello, resulta interesante tratar de completar estos valores a partir de la información intrínseca al *dataset*. Tanto el método MC como el propuesto de reglas asociativas siguen esta máxima. Se ha encontrado que el primero es más rápido, pero el enfoque de reglas asociativas mejora los resultados para los dos clasificadores que han sido analizados, por lo que podemos concluir que este último aprovecha más la información disponible.

Una vez los datos han sido pre-procesados, se ha ejecutado un clasificador SVM. Se han modificado las herramientas disponibles en Weka para optimizar los parámetros del clasificador en función de la medida *g-means*, que suele emplearse para este tipo de problemas. Gracias a este proceso de búsqueda de parámetros, se ha conseguido mejorar la capacidad predictiva del sistema, haciendo patente la flexibilidad de SVM. Sin embargo, este proceso es lento y laborioso, ya que el algoritmo suele tomar bastante tiempo para converger. Se ha conseguido una *g-means* media de 71.64%.

Con propósito comparativo, se ha analizado el comportamiento de un clasificador basado en bosques aleatorios ante el mismo *dataset*, con el mismo preprocesado. Se ha visto que este método es más sencillo de utilizar, pero menos configurable, y no se ha conseguido mejorar el rendimiento de SVM, consiguiendo una *g-means* de 70.37%. Sin embargo, los tiempos de entrenamiento son mucho menores, por lo que sería una opción a considerar si no se dispusiera

de suficiente tiempo para procesar.

Como conclusión, con un *g-means* del 71.64 % podemos decir que si bien el método propuesto no es lo suficientemente preciso como para tomar decisiones de forma autónoma, puede resultar de utilidad a la hora de detectar potenciales impagadores en el momento del alta, de forma que se les de un seguimiento más cercano. En otras palabras, el sistema es suficientemente bueno como para cumplir el objetivo inicial: actuar como sistema de soporte de decisiones.

Hay varias líneas de trabajo futuro, entre las que destacan:

1. Añadir datos de redes sociales al análisis, de forma que se tenga mayor información sobre el cliente en el momento de su alta. Algunos estudios [52] han demostrado que esta información puede ayudar a mejorar la capacidad predictiva del modelo. Dependiendo de los datos empleados, esto podría suponer que el tamaño del *dataset* creciera considerablemente, por lo que habría que recurrir a sistemas diseñados para tratar grandes volúmenes de datos, como Hadoop o Spark.
2. Utilizar el *data warehouse* para analizar el comportamiento de los clientes, dando seguimiento a sus probabilidades de impago y *churn*. Con esto, se aprovechará la infraestructura implementada, ofreciendo aplicaciones muy útiles de cara al negocio: se podría dar especial seguimiento o imponer límites de consumo a aquellos clientes con alto riesgo de impago u ofrecer nuevas ofertas a aquellos cuya probabilidad de *churn* (de marcharse del operador) esté creciendo.
3. Modificar el algoritmo de reglas asociativas para que *class association rules* soporte más métricas, ya que Weka únicamente permite utilizar la confianza. De esta forma, será posible rellenar valores nulos siguiendo diferentes criterios, consiguiendo adaptarse mejor al *dataset*, y profundizando en un método de rellenado de valores incompletos que mejora los comúnmente usados.

Glosario de acrónimos

- **AUC**: Area Under Curve.
- **BI**: Business Intelligence.
- **CAIM**: Class-Attribute Interdependence Maximization.
- **CDR**: Call Detail Register.
- **CRM**: Customer Relationship Manager.
- **CSV**: Comma Separated Values.
- **DDS**: Dimensional Data Store.
- **DM**: Data Mart.
- **DSA**: Data Staging Area.
- **DSS**: Decision Support System.
- **DWH**: Data Warehouse.
- **ELT**: Extract, Load and Transform.
- **ETL**: Extract, Transform and Load.
- **FN**: False Negative.
- **FP**: False Positive.
- **GUI**: Graphical User Interface.
- **ICCID**: Integrated Circuit Card Identifier.
- **IDE**: Integrated Development Environment.
- **IMSI**: International Mobile Subscriber Identity.
- **IP**: Information Package.
- **KDD**: Knowledge Discovery in Databases.
- **MC**: Most Common.
- **MSISDN**: Mobile Subscriber ISDN (Integrated Services for Digital Network) Number.
- **MV**: Missing Values.
- **MVNE**: Mobile Virtual Network Enabler.
- **MVNO**: Mobile Virtual Network Operator.

- **NDS**: Normalized Data Store.
- **OLAP**: OnLine Analytical Processing.
- **OLTP**: OnLine Transaction Processing.
- **RBF**: Radial Basis Function.
- **RDBMS**: Relational Database Management System.
- **REGEX**: Regular Expression.
- **SIM**: Subscriber Identity Module.
- **SMO**: Sequential Minimal Optimization.
- **SMOTE**: Synthetic Minority Over-sampling Technique.
- **SQL**: Structured Query Language.
- **SVM**: Support Vector Machines.
- **TN**: True Negative.
- **TP**: True Positive.

Bibliografía

- [1] Communications Fraud Control Association. Global fraud loss survey 2013. http://www.cvidya.com/media/62059/global-fraud_loss_survey2013.pdf, 2013. Consultado el 20/02/2014.
- [2] Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom. *Database systems: the complete book*. Prentice Hall, Upper Saddle River, NJ, 2008.
- [3] Universidad Politécnica de Valencia. Historia de las bases de datos. <http://histinf.blogs.upv.es/2011/01/04/historia-de-las-bases-de-datos/>, Enero 2011. Consultado el 09/07/2014.
- [4] Frank da Cruz. Columbia university computing history: Herman hollerith. <http://www.columbia.edu/cu/computinghistory/hollerith.html>, Marzo 2011. Consultado el 09/07/2014.
- [5] S. K. Singh. *Database systems concepts, design, and applications*. Dorling Kindersley, New Delhi, India, 2006.
- [6] Russell Dyer. *MySQL in a nutshell*. O'Reilly Media, Sebastopol, CA, 2008.
- [7] Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Jeremy D. Zawodny, Arjen Lentz, and Derek J. Balling. *High performance MySQL*. O'Reilly Media, Sebastopol, CA, 2008.
- [8] Mysql 5.1 reference manual. <http://dev.mysql.com/doc/refman/5.1/en/>, 2008. Consultado el 14/07/2014.
- [9] Flora S. Tsai and Agus T. Kwee. Database optimization for novelty mining of business blogs. *Expert Syst. Appl.*, 38(9):11040–11047, September 2011.
- [10] Rackspace Support. Mysql engines - myisam vs innodb. http://www.rackspace.com/knowledge_center/article/mysql-engines-myisam-vs-innodb. Consultado el 15/07/2014.
- [11] Marco de desarrollo de la junta de andalucía. http://www.juntadeandalucia.es/servicios/madeja/contenido/libro-pautas/18#Tipo_de_las_variables_para_uso_monetario. Consultado el 16/07/2014.
- [12] Oracle. Mapping sql and java types. <http://docs.oracle.com/javase/1.5.0/docs/guide/jdbc/getstart/mapping.html>, 1999. Consultado el 16/07/2014.
- [13] W1#1872: Add support for dates from 2038 till 2116 as values of timestamp type. <http://dev.mysql.com/worklog/task/?id=1872>. Consultado el 22/07/2014.
- [14] Oltp vs. olap. <http://datawarehouse4u.info/OLTP-vs-OLAP.html>, 2009. Consultado el 25/04/2015.
- [15] Paulraj Ponniah. *FUNDAMENTALS DATA WAREHOUSING FUNDAMENTALS A Comprehensive Guide for IT Professionals*, volume 6. Wiley, 2001.

- [16] Ralph Kimball and Margy Ross. The data warehouse toolkit: the complete guide to dimensional modelling. *Nachdr.]. New York [ua]: Wiley*, pages 1–447, 2002.
- [17] Robert J Davenport. Etl vs elt. <http://www.dataacademy.com/files/ETL-vs-ELT-White-Paper.pdf>, 2008. Consultado el 26/04/2015.
- [18] Shaker H. Ali El-Sappagh, Abdeltawab M. Ahmed Hendawi, and Ali Hamed El Bastawissy. A proposed model for data warehouse ETL processes. *Journal of King Saud University - Computer and Information Sciences*, 23(2):91–104, 2011.
- [19] Vincent Rainardi. *Building a Data Warehouse*. APress, 2008.
- [20] Thomas C. Hammergren. *Official Sybase Data Warehousing on the Internet: Accessing the Corporate Knowledge Base (How to Guides)*. Coriolis Group, 1998.
- [21] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data mining practical machine learning tools and techniques*. Morgan Kaufmann, Burlington, MA, 2011.
- [22] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Addison Wesley, Boston, 2005.
- [23] Andrew Ng. Cs229 lecture notes. <http://cs229.stanford.edu/>, 2007. Consultado el 24/05/2015.
- [24] Dorian Pyle. *Data Preparation for Data Mining (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1999.
- [25] Soumen Chakrabarti. *Data mining : know it all*. Elsevier/Morgan Kaufmann Publishers, Burlington, MA, 2009.
- [26] Julián Luengo, Salvador García, and Francisco Herrera. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and Information Systems*, 32(1):77–108, 2012.
- [27] Sean Borman. The expectation maximization algorithm: A short tutorial, 2004.
- [28] Xiaoyuan Su, Taghi M. Khoshgoftaar, and Russell Greiner. Using imputation techniques to help learn accurate classifiers. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, nov 2008.
- [29] J Dougherty, R Kohavi, and M Sahami. Supervised and unsupervised discretization of continuous features. *Machine Learning: Proceedings of the Twelfth International Conference*, 54(2):194–202, 1995.
- [30] Usama M. Fayyad and Keki B. Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning, 1993.
- [31] Igor Kononenko. On Biases in Estimating Multi-Valued Attributes. In *IJCAI*, pages 1034–1040, 1995.
- [32] Randy Kerber. Chimerge: Discretization of numeric attributes. *Proceedings of the tenth national conference on Artificial intelligence*, pages 123–128, 1992.
- [33] Lukasz Kurgan and Krzysztof J. Cios. Discretization algorithm that uses class-attribute interdependence maximization. In *Proc. of the 2001 International Conference on Artificial Intelligence (ICAI-2001)*, pages 980–987, 2001.

- [34] Alberto Cano, Dat T. Nguyen, Sebastián Ventura, and Krzysztof J. Cios. ur-CAIM: improved CAIM discretization for unbalanced and balanced data. *Soft Computing*, 2014.
- [35] Qiusha Zhu, Lin Lin, Mei Ling Shyu, and Shu Ching Chen. Effective supervised discretization for classification based on correlation maximization. In *Proceedings of the 2011 IEEE International Conference on Information Reuse and Integration, IRI 2011*, pages 390–395, 2011.
- [36] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying Support Vector Machines to Imbalanced Datasets. *Machine Learning: ECML 2004*, 3201:39–50, 2004.
- [37] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMO-TE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
- [38] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A Practical Guide to Support Vector Classification. *BJU international*, 101(1):1396–400, 2008.
- [39] Nokia Siemens Networks. Mobile virtual network operator. http://br.networks.nokia.com/sites/default/files/MVNO_A4_2106.pdf, 2007. Consultado el 06/06/2015.
- [40] Ley 25/2007, de 18 de octubre, de conservación de datos relativos a las comunicaciones electrónicas y a las redes públicas de comunicaciones. http://noticias.juridicas.com/base_datos/Admin/125-2007.html#a3. Consultado el 13/06/2015.
- [41] neustar. What the fraud? a look at telecommunications fraud and its impacts. <https://www.neustar.biz/resources/whitepapers/csp/telecom-fraud-impacts-ebook.pdf>. Consultado el 20/06/2015.
- [42] Gary Weiss. *Encyclopedia of Data Warehousing and Mining, Second Edition*. IGI Global, 2009.
- [43] John Shawe-Taylor, Keith Howker, and Peter Burge. Detection of fraud in mobile telecommunications. *Information Security Technical Report*, 4:8–9, jan 1999.
- [44] K.J. Ezawa and S.W. Norton. Constructing bayesian networks to predict uncollectible telecommunications accounts. *IEEE Expert*, 11(5):45–51, Oct 1996.
- [45] S. Daskalaki, I. Kopanas, M. Goudara, and N. Avouris. Data mining for decision support on customer insolvency in telecommunications business. *European Journal of Operational Research*, 145(2):239–255, 2003.
- [46] Carlos André R. Pinheiro, Alexandre G. Evsukoff, and Nelson F. F. Ebecken. Revenue recovering with insolvency prevention on a brazilian telecom operator. *SIGKDD Explor. Newsl.*, 8(1):65–70, June 2006.
- [47] Tomasz S. Ząbkowski and Wiesław Szczesny. Insolvency modeling in the cellular telecommunication industry. *Expert Systems with Applications*, 39(8):6879–6886, 2012.
- [48] Roselina Sallehuddin, Subariah Ibrahim, Abdikarim Hussein Elmi, et al. Classification of sim box fraud detection using support vector machine and artificial neural network. *International Journal of Innovative Computing*, 4(2), 2014.
- [49] Remco R Bouckaert, Eibe Frank, Mark Hall, Richard Kirkby, Peter Reutemann, Alex Seewald, and David Scuse. WEKA Manual for Version 3-7-12. *Interface*, 2014.

- [50] How do i divide a dataset into training and test set? <https://weka.wikispaces.com/How+do+I+divide+a+dataset+into+training+and+test+set%3F>, 2009. Consultado el 14/07/2015.
- [51] Classbalancer vs. costsensitiveclassifier. <http://weka.8497.n7.nabble.com/ClassBalancer-vs-CostSensitiveClassifier-td34687.html>, 2015. Consultado el 14/07/2015.
- [52] Management Solutions. Data science and the transformation of the financial industry. <http://www.managementsolutions.com/PDF/ENG/datascience-eng.pdf>, 2015. Consultado el 25/10/2015.

Anexos



Código empleado

A.1. *User defined java class*: Encontrar tipo documento REGEX

Código A.1: *User defined java class* Encontrar tipo documento REGEX

```
1 import java.util.regex.Pattern;
2 import java.util.regex.Matcher;
3
4 private String id;
5
6 // NIF pattern
7 private Pattern NIFpattern;
8
9 // CIF pattern
10 private Pattern CIFpattern;
11
12 // NIE pattern
13 private Pattern NIEpattern;
14
15 private boolean IsNIF()
16 {
17     // Check if the string introduced is a NIF
18     Matcher matcher = NIFpattern.matcher(id);
19     return (matcher.find());
20 }
21
22 private boolean IsCIF()
23 {
24     // Check if the string introduced is a CIF
25     Matcher matcher = CIFpattern.matcher(id);
26     return (matcher.find());
27 }
28
```

```

29 private boolean IsNIE()
30 {
31     // Check if the string introduced is a NIE
32     Matcher matcher = NIEpattern.matcher(id);
33     return (matcher.find());
34 }
35
36 public boolean processRow(StepMetaInterface smi, StepDataInterface sdi) throws
    ↪ KettleException
37 {
38     Object[] r = getRow();
39
40     if (r == null)
41     {
42         setOutputDone();
43         return false;
44     }
45
46     if (first)
47         first = false;
48
49     r = createOutputRow(r, data.outputRowMeta.size());
50     id = get(Fields.In, "Documento").getString(r);
51
52
53     if (id == null)
54         get(Fields.Out, "Tipo_Documento").setValue(r, "UNK");
55
56     else
57     {
58         // Return the corresponding type
59         // Check the NIF pattern
60         NIFpattern = Pattern.compile("\\d{8}[A-Z]");
61         if (IsNIF())
62         {
63             get(Fields.Out, "Tipo_Documento").setValue(r, "NIF");
64             // Send the row on to the next step.
65             putRow(data.outputRowMeta, r);
66             return true;
67         }
68
69         // Check the NIE pattern
70         NIEpattern = Pattern.compile("[XYZ]\\d{7}[A-Z]");
71         if (IsNIE())
72         {
73             get(Fields.Out, "Tipo_Documento").setValue(r, "NIE");
74             // Send the row on to the next step.
75             putRow(data.outputRowMeta, r);
76             return true;
77         }
78

```

```

79     // Check the CIF pattern
80     CIFpattern = Pattern.compile("[A-W]\\d{7}[A-Z[\\d]]");
81     if (IsCIF()) {
82         get(Fields.Out, "Tipo_Documento").setValue(r, "CIF");
83         // Send the row on to the next step.
84         putRow(data.outputRowMeta, r);
85         return true;
86     }
87
88     get(Fields.Out, "Tipo_Documento").setValue(r, "UNK");
89
90 }
91 // Send the row on to the next step.
92 putRow(data.outputRowMeta, r);
93
94 return true;
95 }

```

A.2. *Execute SQL script* Populate tables

Código A.2: Sentencias INSERT INTO... SELECT FROM ... en el paso *Execute SQL script* Populate tables

```

1 drop table if exists rm_bi_dds_impago._temp_fact_ant;
2 drop table if exists rm_bi_dds_impago._temp_resumen_pp;
3
4 insert into dim_comunidad_autonoma
5 select * from rm_bi_nds.info_comunidad_autonoma;
6
7 insert into dim_provincia
8 select p.info_provincia_id,
9 p.provincia,
10 p.info_comunidad_autonoma_id,
11 p.provincia_walva
12 from rm_bi_nds.info_provincia p
13 order by p.info_provincia_id;
14
15 insert into dim_pais
16 select * from rm_bi_nds.info_nacionalidad;
17
18 insert into dim_tipo_cliente
19 select * from rm_bi_nds.info_tipo_cliente;
20
21 insert into dim_tipo_documento
22 select * from rm_bi_nds.info_tipo_documento;
23
24 insert into dim_linea_estado
25 select * from rm_bi_nds.cliente_linea_estado;

```

A.3. *Table input* Table input cliente

Código A.3: Sentencia SELECT empleada en la transformación fact_cliente

```
1 SELECT
2 cd.id_cliente,
3 cd.nombre_apellidos,
4 if(cd.sexo='F', 1, 0) as sexo_fem,
5 cd.info_nacionalidad_id,
6 year(curdate())-year(cd.fecha_nacimiento) as edad,
7 cd.info_tipo_documento_id,
8 cd.documento,
9 substring_index(lower(Email),'@',-1) as proveedor_email,
10 substring_index(lower(Email),'.',-1) as dominio_email,
11 cd.emailing,
12 cd.cliente_estado_id,
13 cd.roaming_activo_cliente,
14 cd.info_tipo_cliente_id,
15 cd.fecha_alta,
16 cd.info_provincia_id,
17 cd.direccion,
18 cd.envio_direccion,
19 cd.cliente_datos_banco_id,
20 cd.num_interacciones,
21 count(distinct cl.msisdn) as num_lineas,
22 sum(case when cl.tipo_alta = 'PORTING' and cl.tipo_contrato_anterior = 'P' then
    ↪ 1 else 0 end) as num_porta_pre,
23 sum(case when cl.tipo_alta = 'PORTING' and cl.tipo_contrato_anterior = 'T' then
    ↪ 1 else 0 end) as num_porta_post,
24 sum(case when cl.tipo_alta = 'NEWLINE' then 1 else 0 end) as num_nuevas,
25 sum(case when cl.info_metodo_pago_id = 1 then 1 else 0 end) as num_pago_pdv,
26 sum(case when cl.info_metodo_pago_id = 2 then 1 else 0 end) as num_pago_bank,
27 sum(case when cl.info_metodo_pago_id = 3 then 1 else 0 end) as num_pago_tpv,
28 sum(case when cl.info_tipo_sim_id = 1 then 1 else 0 end) as num_sim,
29 sum(case when cl.info_tipo_sim_id = 2 then 1 else 0 end) as num_usim,
30 sum(case when cl.info_producto_id in (2,9) then 1 else 0 end) as num_prod_mini,
31 sum(case when cl.info_producto_id in (1,8) then 1 else 0 end) as num_prod_peque,
32 sum(case when cl.info_producto_id in (6,7) then 1 else 0 end) as
    ↪ num_prod_peque100,
33 sum(case when cl.info_producto_id in (4,10) then 1 else 0 end) as
    ↪ num_prod_mediana,
34 sum(case when cl.info_producto_id in (3,11) then 1 else 0 end) as
    ↪ num_prod_grande,
35 sum(case when cl.info_producto_id in (7,8,9,10,11) then 1 else 0 end) as
    ↪ num_prod_plus
36 FROM rm_bi_nds.cliente_datos cd
37 inner join rm_bi_nds.cliente_linea cl on cd.id_cliente = cl.id_cliente and cl.
    ↪ date_to = '2199-12-31 23:59:59'
38 where cd.date_to = '2199-12-31 23:59:59'
39 group by cd.id_cliente
```

A.4. *Execute SQL script* _temp_fact_ant y _temp_resumen_pp

Código A.4: Sentencias CREATE en el paso _temp_fact_ant y _temp_resumen_pp

```
1 drop table if exists rm_bi_dds_impago._temp_fact_ant;
2 drop table if exists rm_bi_dds_impago._temp_resumen_pp;
3
4 create table IF NOT EXISTS rm_bi_dds_impago._temp_fact_ant
5 (PRIMARY KEY (id_cliente, periodo_siguiente))
6 select
7 f_ant.id_cliente,
8 f_ant.periodo_factura + interval 1 month as periodo_siguiente,
9 f_ant.importe_recibo as importe_recibo_ant,
10 f_ant.estado_cobro
11 from rm_bi_nds.factura_cliente f_ant
12 where f_ant.date_to = '2199-12-31 23:59:59';
13
14 create table IF NOT EXISTS rm_bi_dds_impago._temp_resumen_pp
15 (PRIMARY KEY (factura_cliente_id))
16 select fcp.factura_cliente_id,
17 sum(fcp.num_descendientes) as pp_descendientes,
18 sum(fcp.importe) as pp_importe
19 from
20 rm_bi_nds.factura_cliente_pioneros fcp
21 where fcp.date_to = '2199-12-31 23:59:59'
22 group by fcp.factura_cliente_id;
```

A.5. *Table input* en la transformación fact_factura

Código A.5: Sentencia SELECT empleada en la transformación fact_factura

```
1 select
2 c.fact_cliente_id,
3 df.dim_fecha_id as dim_fecha_id_periodo_fact,
4 f.num_factura,
5 ifnull(fp.pp_importe, 0) as pp_importe,
6 f.imp_total_plan_pionero,
7 f.imp_factura_plan_pionero,
8 f.imp_ingreso,
9 f.importe_recibo_cobrado,
10 f.importe_recibo,
11 f.imp_iva,
12 f.imp_total_antes_impuestos,
13 ifnull(fp.pp_descendientes, 0) as pp_num_descendientes,
14 if(f.estado_cobro=2, 1, 0) as impago_periodo,
15 if(fant.estado_cobro=2, 1, 0) as impago_periodo_anterior,
16 if(fant.importe_recibo_ant is not null and fant.importe_recibo_ant > 0.01, round
    ↪ (f.importe_recibo / fant.importe_recibo_ant, 4), 0) as
    ↪ dif_importe_anterior,
17 (case when fpro.cod_promo = 'HM712' then fpro.cantidad else 0 end) as num_promo,
```

```

18 (case when fpro.cod_promo = 'HM713' then fpro.cantidad else 0 end) as
    ↪ num_compensaciones,
19 if(round(abs(f.imp_iva + f.imp_total_antes_impuestos - f.importe_recibo),2)
    ↪ >0.01, 1, 0) as alarma_descuadre1,
20 if(round(abs(f.imp_total_plan_pionero - f.imp_factura_plan_pionero - imp_ingreso
    ↪ ),2)>0.01, 1, 0) as alarma_descuadre2,
21 if(round(abs(f.imp_total_plan_pionero - ifnull(fp.pp_importe, 0)),2)>0.01, 1, 0)
    ↪ as alarma_descuadre3
22 from rm_bi_nds.factura_cliente f
23 /*tomar id de fact table de clientes en el momento de la factura*/
24 inner join rm_bi_dds_impago.fact_cliente c on c.id_cliente = f.id_cliente
25     and DATE_SUB(f.Fecha_emision,INTERVAL 1 DAY) between c.date_from and c.
    ↪ date_to
26 inner join rm_bi_dds_impago.dim_fecha df on df.fecha = f.periodo_factura
27 left join rm_bi_dds_impago._temp_resumen_pp fp on fp.factura_cliente_id = f.id
28 left join rm_bi_dds_impago._temp_fact_ant fant
29     on fant.id_cliente = f.id_cliente and fant.periodo_siguiente = f.
    ↪ periodo_factura
30 left join rm_bi_nds.factura_promocion fpro on fpro.factura_cliente_id = f.id
31     and fpro.cod_promo <> 'HM711'
32     and fpro.date_to = '2199-12-31 23:59:59'
33 where f.date_to = '2199-12-31 23:59:59'
34     /*no interesa aquello que no se ha intentado cobrar*/
35     and f.estado_cobro <> 0
36 group by f.id

```

A.6. *Table input* Input DDS de la transformación weka_cliente

Código A.6: Sentencia SELECT empleada en la transformación weka_cliente

```

1
2 SELECT
3 msisdn,
4 tipo_documento,
5 genero,
6 ifnull(TIMESTAMPDIFF(YEAR, cd.fecha_nacimiento,CURDATE()),'??') as edad,
7 codigo_continente,
8 cod_nacionalidad,
9 comunidad_autonoma,
10 provincia,
11 proveedor_email,
12 dominio_email,
13 permite_email,
14 sim_titular_banco_linea,
15 sim_direccion_envio_linea,
16 rango_sim_titular_banco_linea,
17 rango_sim_direccion_envio_linea,
18 tipo_cliente,
19 dp.prod_codigo,
20 dp.prod_throttle,

```

```

21 dp.prod_minutos,
22 dp.prod_megas + dp.prod_megas_extra as prod_megas,
23 dp.prod_precio,
24 dol.operador_origen,
25 dol.tipo_alta,
26 dol.tipo_contrato_origen,
27 dm.tipo_sim,
28 dm.tipo_pago,
29 dpp.tiene_padrino,
30 dim_num_msisdn_ant as num_msisdn_ant,
31 dim_dias_msisdn_ant as dias_msisdn_ant,
32 ifnull(dcan.canal_nombre, 'Televenta República Móvil') as canal_nombre,
33 ifnull(dcan.canal_televenta, 'Televenta') as canal_televenta,
34 if(ff.impagadas>1, 'Impagador', 'Pagador') as impagador
35 FROM fact_linea fl
36 inner join dim_cliente dc on fl.dim_cliente_id = dc.dim_cliente_id
37 inner join dim_misc dm on fl.dim_misc_id = dm.dim_misc_id
38 inner join dim_producto dp on dp.dim_producto_id = fl.dim_producto_id
39 inner join dim_origen_linea dol on dol.dim_origen_linea_id = fl.
    ↪ dim_origen_linea_id
40 left join dim_canal dcan on dcan.dim_canal_id = fl.dim_canal_id
41 inner join dim_pioneros dpp on dpp.dim_pioneros_id = fl.dim_pioneros_id
42 inner join dim_fecha df on df.dim_fecha_id = fl.dim_fecha_activacion_id
43 inner join _temp_estado_fact ff on ff.id_cliente = dc.id_cliente
44 left join rm_bi_nds.cliente_datos cd on cd.id_cliente = dc.id_cliente and cd.
    ↪ date_to > curdate()
45 where (impagadas > 0 or cobradas > 0)
46 and fl.date_from = '1900-01-01 00:00:00'
47 order by fl.msisdn

```

A.7. *Modified Java Script Value* Missing values -> ? de la transformación weka_cliente

Código A.7: *Modified Java Script Value* Missing values -> ? de la transformación weka_cliente, basado en el código del usuario RFVoltolini en <http://stackoverflow.com/questions/17464271/how-to-recode-many-fields-in-kettle-all-of-which-require-the-same-recoding>

```

1 //Script here
2 // list of fields you wish to map
3 var fieldsToMap = ["tipo_documento",
4 "genero",
5 "edad",
6 "codigo_continente",
7 "cod_nacionalidad",
8 "comunidad_autonoma",
9 "provincia",
10 "proveedor_email",
11 "dominio_email",
12 "permite_email",

```

```

13 "rango_sim_titular_banco_linea",
14 "rango_sim_direccion_envio_linea",
15 "tipo_cliente",
16 "prod_codigo",
17 "prod_throttle",
18 "tipo_alta",
19 "tipo_sim",
20 "tipo_pago",
21 "tiene_padrino",
22 "canal_nombre",
23 "canal_televenta",
24 "impagador",
25 "origen_linea_w"];
26 var tmpField;
27 var fieldIndex;
28
29 // for each field to map...
30 for (var i=0; i < fieldsToMap.length; i++) {
31     //get the index of the field once you only have it's name
32     fieldIndex = getInputRowMeta().indexOfValue(fieldsToMap[i]);
33     //get the field
34     tmpField = row.getValue(fieldIndex);
35
36     if (fieldsToMap[i] == "edad" && tmpField < 5)
37         tmpField.setValue("?");
38
39     //don't forget to trim as Kettle usually pads strings
40     switch (trim(tmpField)) {
41         case "Desconocido":
42             tmpField.setValue("?");
43             break;
44         case "Desconocida":
45             tmpField.setValue("?");
46             break;
47     }
48 }

```

A.8. Programa Java para la evaluación de las SVM

Código A.8: evaluate_smo.java

```

1 package smo_cliente;
2
3 import java.io.BufferedWriter;
4 import java.io.File;
5 import java.io.FileWriter;
6 import java.io.IOException;
7 import java.text.SimpleDateFormat;
8 import java.util.Date;
9 import java.util.Vector;

```

```

10
11 import weka.core.Instances;
12 import weka.core.SerializationHelper;
13 import weka.core.converters.CSVLoader;
14 import weka.core.setupgenerator.Point;
15 import weka.filters.Filter;
16 import weka.filters.unsupervised.instance.Randomize;
17 import weka.filters.unsupervised.instance.RemovePercentage;
18 import weka.filters.unsupervised.attribute.Remove;
19 import weka.filters.unsupervised.attribute.ReplaceMissingValues;
20 import weka.filters.supervised.attribute.Discretize;
21 import weka.filters.supervised.instance.SMOTE;
22 import weka.filters.supervised.instance.ClassBalancer;
23 import weka.classifiers.Classifier;
24 import weka.classifiers.evaluation.Evaluation;
25 import weka.classifiers.evaluation.output.prediction.CSV;
26 import weka.classifiers.functions.SMO;
27 import weka.classifiers.meta.MultiSearch;
28
29 public class evaluate_smo {
30
31     //Instances data;
32
33     public static Instances load_csv (String[] args) throws IOException
34     {
35         CSVLoader loader = new CSVLoader();
36         loader.setSource(new File(args[0]));
37         return loader.getDataSet();
38     }
39
40     public static Instances pre_process_train (Instances input_data)
41     {
42         Instances current_data = null;
43
44         /* The last index contains the class*/
45         input_data.setClassIndex(input_data.numAttributes() - 1);
46
47         String[] options = null;
48         int smote_percentage = 100;
49
50         /* Oversample using SMOTE */
51         System.out.println("Oversampling at "+smote_percentage+"%...");
52         SMOTE oversample = new SMOTE();
53         try {
54             options = weka.core.Utils.splitOptions("-C 0 -K 5 -P "+
55                 ↵ smote_percentage+" -S 1");
56             oversample.setOptions(options);
57             oversample.setInputFormat(input_data);
58             current_data = Filter.useFilter(input_data, oversample);
59         } catch (Exception e) {
60             e.printStackTrace();
61         }
62     }
63 }

```

```

60         return null;
61     }
62
63     /* The last index contains the class*/
64     current_data.setClassIndex(current_data.numAttributes() - 1);
65
66     /* Apply weighting */
67     System.out.println("Applying weighting...");
68     ClassBalancer cb = new ClassBalancer();
69     try {
70         cb.setInputFormat(current_data);
71         current_data = Filter.useFilter(current_data, cb);
72     } catch (Exception e) {
73         e.printStackTrace();
74         return null;
75     }
76
77     /*fill missing values using Mean/Mode*/
78     System.out.println("Replacing missing values...");
79     ReplaceMissingValues replace = new ReplaceMissingValues();
80     try {
81         replace.setInputFormat(current_data);
82         current_data = Filter.useFilter(current_data, replace);
83     } catch (Exception e) {
84         e.printStackTrace();
85         return null;
86     }
87
88
89     System.out.println("Train pre-processing done.");
90     return current_data;
91 }
92
93
94 public static Instances pre_process_test (Instances input_data)
95 {
96     Instances current_data = null;
97
98     /* The last index contains the class*/
99     input_data.setClassIndex(input_data.numAttributes() - 1);
100
101     /*fill missing values using Mean/Mode*/
102     System.out.println("Replacing missing values...");
103     ReplaceMissingValues replace = new ReplaceMissingValues();
104     try {
105         replace.setInputFormat(input_data);
106         current_data = Filter.useFilter(input_data, replace);
107     } catch (Exception e) {
108         e.printStackTrace();
109         return null;
110     }

```

```

111         System.out.println("Test pre-processing done.");
112         return current_data;
113     }
114 }
115
116 public static Instances pre_process_common (Instances input_data)
117 {
118     Instances current_data = null;
119
120     /* The last index contains the class*/
121     input_data.setClassIndex(input_data.numAttributes() - 1);
122
123     /*delete useless attributes*/
124     System.out.println("Removing unwanted attributes...");
125     Remove remove = new Remove();
126     String[] options = null;
127
128     try {
129         options = weka.core.Utils.splitOptions("-R 13-14");
130         remove.setOptions(options);
131         remove.setInputFormat(input_data);
132         current_data = Filter.useFilter(input_data, remove);
133     } catch (Exception e) {
134         e.printStackTrace();
135         return null;
136     }
137
138     current_data.setClassIndex(current_data.numAttributes() - 1);
139
140     /* Discretize using Fayyad, Irani's method */
141     /*
142     * This should not be a common process as it would imply a "leak
143     * ↔ from the future".
144     * However, it's easier to assume that the discretization
145     * ↔ boundaries would be
146     * the ones defined in this phase.
147     */
148     System.out.println("Discretizing...");
149     Discretize discretize = new Discretize();
150     try {
151         options = weka.core.Utils.splitOptions("-R first-last -
152         ↔ precision 6");
153         discretize.setOptions(options);
154         discretize.setInputFormat(current_data);
155         current_data = Filter.useFilter(current_data, discretize);
156     } catch (Exception e) {
157         e.printStackTrace();
158         return null;
159     }

```

```

159
160     System.out.println("Common pre-processing done.");
161     return current_data;
162
163 }
164
165 public static SMO train_classifier (Instances data, double C, double
    ↪ gamma)
166 {
167
168     SMO svm = new SMO();
169     String[] options;
170     try {
171         options = weka.core.Utils.splitOptions("-L 0.001 -P 1.0E
    ↪ -12 -N 0 -V -1 -W 1 -K \"weka.classifiers.functions
    ↪ .supportVector.RBFKernel -G "+ gamma +" -C 250007\"
    ↪ ");
172         svm.setOptions(options);
173         svm.setC(C);
174         svm.setBuildLogisticModels(true);
175         System.out.println("Running SMO:");
176         System.out.println("C: " + svm.getC());
177         System.out.println("Kernel: " + svm.getKernel());
178         System.out.println("Epsilon: " + svm.getEpsilon());
179         System.out.println("Filters: " + svm.getFilterType());
180         svm.buildClassifier(data);
181     } catch (Exception e) {
182         e.printStackTrace();
183         return null;
184     }
185
186     return svm;
187 }
188
189 public static void main(String[] args) throws Exception
190 {
191     /* A CSV file must be defined in an input argument */
192     Instances data = load_csv (args);
193
194     Instances filtered_data = null;
195     filtered_data = pre_process_common(data);
196
197     /* Define ten seeds to be able to reproduce results later (if
    ↪ needed) */
198     int [] seedArray = {8, 20, 23, 32,49,58, 60, 63, 70, 97};
199
200     for(int nExec=0; nExec < seedArray.length; nExec++)
201     {
202         /* Get date for information purposes */
203         Date date = new Date();

```

```

204 SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy h:
    ↪ mm:ss");
205 String formattedDate = sdf.format(date);
206 System.out.println("(" + formattedDate + ") Execution #" +
    ↪ nExec + ". Seed = " + rnd.getRandomSeed());
207
208 /*Split dataset into test and train*/
209 Randomize rnd = new Randomize();
210 int seed = seedArray[nExec];
211 rnd.setRandomSeed(seed);
212 rnd.setInputFormat(filtered_data);
213 filtered_data = Filter.useFilter(filtered_data, rnd);
214 RemovePercentage rp = new RemovePercentage();
215 rp.setInputFormat(filtered_data);
216 rp.setPercentage(33.33);
217 rp.setInvertSelection(true);
218 Instances test_data = Filter.useFilter(filtered_data, rp);
219 rp.setInvertSelection(false);
220 rp.setInputFormat(filtered_data);
221 rp.setPercentage(33.33);
222 Instances train_data = Filter.useFilter(filtered_data, rp)
    ↪ ;
223 System.out.println("Test samples: " + test_data.
    ↪ numInstances());
224 System.out.println("Train samples: " + train_data.
    ↪ numInstances());
225
226 /*Find best parameters for this particular train set*/
227 train_data = pre_process_train (train_data);
228 train_data.setClassIndex(train_data.numAttributes() - 1);
229
230 MultiSearch ms = new MultiSearch();
231 String[] options;
232 /* Parameters for first grid search */
233 options = weka.core.Utils.splitOptions("-E GM -search \"
    ↪ weka.core.setupgenerator.MathParameter -property
    ↪ classifier.c -min -1.0 -max 3.0 -step 2.0 -base 2.0
    ↪ -expression pow(BASE,I)\" -search \"weka.core.
    ↪ setupgenerator.MathParameter -property classifier.
    ↪ kernel.gamma -min -7.0 -max -3.0 -step 2.0 -base
    ↪ 2.0 -expression pow(BASE,I)\" -sample-size 100.0 -
    ↪ log-file C:\\datos\\test_new_\"+seed +\".txt -initial
    ↪ -folds 3 -subsequent-folds 0 -num-slots 4 -S 1 -W
    ↪ weka.classifiers.functions.SMO -- -C 1.0 -L 0.001 -
    ↪ P 1.0E-12 -N 0 -V -1 -W 1 -K \"weka.classifiers.
    ↪ functions.supportVector.RBFKernel -G 0.01 -C
    ↪ 250007\"");
234
235 /* In case more values for C are needed*/
236 //options = weka.core.Utils.splitOptions("-E GM -search \"
    ↪ weka.core.setupgenerator.MathParameter -property

```

```

237     ↪ classifier.c -min -1.0 -max 3.0 -step 2.0 -base 2.0
238     ↪ -expression pow(BASE,I)\ " -sample-size 100.0 -log-
239     ↪ file C:\\datos\\test_new_"+seed+".txt -initial-
     ↪ folds 3 -subsequent-folds 0 -num-slots 4 -S 1 -W
     ↪ weka.classifiers.functions.SMO -- -C 1.0 -L 0.001 -
     ↪ P 1.0E-12 -N 0 -V -1 -W 1 -K \"weka.classifiers.
     ↪ functions.supportVector.RBFKernel -G 0.5 -C
     ↪ 250007\");

/*In case more values for gamma are needed*/
//options = weka.core.Utils.splitOptions("-E GM -search \"
     ↪ weka.core.setupgenerator.MathParameter -property
     ↪ classifier.kernel.gamma -min -7.0 -max -3.0 -step
     ↪ 2.0 -base 2.0 -expression pow(BASE,I)\ " -sample-
     ↪ size 100.0 -log-file C:\\datos\\test_new_assoc"+
     ↪ seed+".txt -initial-folds 3 -subsequent-folds 0 -
     ↪ num-slots 4 -S 1 -W weka.classifiers.functions.SMO
     ↪ -- -C 32 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"
     ↪ weka.classifiers.functions.supportVector.RBFKernel
     ↪ -G 0.01 -C 250007\");

240
241     ms.setOptions(options);
242     ms.buildClassifier(train_data);
243
244     /* MultiSearch returns the best classifier, as well as the
     ↪ optimal parameters */
245     Point<Object> best = ms.getValues();
246     System.out.println("C=2^"+(double)best.getValue(0));
247     System.out.println("gamma=2^"+(double)best.getValue(1));
248     Classifier best_cl = ms.getBestClassifier();
249     best_cl.buildClassifier(train_data);
250
251     /* Save model and header, in case further analysis is
     ↪ needed */
252     String filename;
253     filename = "svm_assoc_" + seed + ".model";
254     System.out.println("Saving model in " + filename + "...");
255     Vector v = new Vector();
256     v.add(best_cl);
257     v.add(new Instances(train_data, 0));
258     SerializationHelper.write(filename, v);
259
260     /* Evaluate the classifier after preprocessing test data*/
261     test_data = pre_process_test (test_data);
262     Evaluation eval = new Evaluation(train_data);
263     eval.evaluateModel(best_cl, test_data);
264
265     /* Compute the components for g-means */
266     Double pos_acc = eval.numTruePositives(1)/(eval.
     ↪ numTruePositives(1) + eval.numFalseNegatives(1));

```

```

267         Double neg_acc = eval.numTrueNegatives(1)/(eval.
           ↪ numTrueNegatives(1) + eval.numFalsePositives(1));
268
269         /* Print the results */
270         System.out.println(eval.toSummaryString("\nResults\n
           ↪ =====\n", false));
271         System.out.println("True positives: " + eval.
           ↪ numTruePositives(1));
272         System.out.println("False negatives: " + eval.
           ↪ numFalseNegatives(1));
273         System.out.println("False positives: " + eval.
           ↪ numFalsePositives(1));
274         System.out.println("True negatives: " + eval.
           ↪ numTrueNegatives(1));
275         System.out.println("Positive accuracy: "
           ↪ + pos_acc);
276         System.out.println("Negative accuracy: "
           ↪ + neg_acc);
277         System.out.println("g-means: "
           ↪ + Math.sqrt(pos_acc*neg_acc));
280         System.out.println("AuC: " + eval.areaUnderROC(0));
281         System.out.println(eval.toMatrixString());
282         System.out.println("");
283
284     }
285 }
286
287
288 }

```

A.9. Programa Java para completar datos desconocidos mediante Reglas Asociativas

Código A.9: assoc_rules.java

```

1
2 package smo_cliente;
3 import java.io.File;
4 import java.util.ArrayList;
5 import java.util.HashMap;
6 import java.util.Iterator;
7 import java.util.List;
8 import java.util.Map;
9 import java.util.Collections;
10 import java.util.Comparator;
11 import java.util.Set;
12 import java.util.Map.Entry;
13 import weka.associations.Apriori;
14 import weka.associations.ItemSet;
15 import weka.core.AttributeStats;
16 import weka.core.Instance;

```

```

17 import weka.core.Instances;
18 import weka.core.converters.CSVSaver;
19 import weka.filters.Filter;
20 import weka.filters.supervised.attribute.Discretize;
21
22 public class assoc_rules {
23
24     private static boolean DEBUG = true;
25
26     public static Instances fillMV (Instances data, int attIndex, List<Integer>
        ↪ instMis) throws Exception
27     {
28         /* Run association rules for every class with MV */
29         Apriori AP = new Apriori();
30         /*Confidence - a threshold of 0.9 works better*/
31         String[] options = weka.core.Utils.splitOptions("-N 50 -T 0 -C 0.9 -D
        ↪ 0.05 -U 1.0 -M 0.5 -S -1.0 -A -c " + attIndex);
32         //String[] options = weka.core.Utils.splitOptions("-N 50 -T 0 -C 0.5 -D
        ↪ 0.05 -U 1.0 -M 0.5 -S -1.0 -A -c " + attIndex);
33
34         AP.setOptions(options);
35         AP.buildAssociations(data);
36         if(DEBUG) System.out.println("Found " + AP.toString() + "rules.");
37
38         /* Get all rules */
39         ArrayList<Object>[] m_allTheRules = AP.getAllTheRules();
40         boolean done = false;
41         boolean not_this_rule = false;
42         int a_rule = -1;
43         int idx = 0;
44         int a_inst=-1;
45         int at = 0;
46         List<Integer> temp = new ArrayList<Integer>();
47         ItemSet lstAt;
48         int[] lsa;
49         int numAt = data.numAttributes();
50         int numFilled = 0;
51
52         /* For every missing value of that attribute, try to fill */
53         for(Iterator<Integer> i = instMis.iterator(); i.hasNext(); )
54         {
55             done = false;
56             idx = i.next();
57             Instance inst = data.get(idx);
58             if(DEBUG) System.out.println(inst.toString());
59
60             for(int r = 0; r < m_allTheRules[0].size() && done == false; r++)
61             {
62                 not_this_rule = false;
63                 lstAt = (ItemSet)m_allTheRules[0].get(r);
64                 lsa = lstAt.items();

```

```

65
66     temp.clear();
67     for(int h=0; h<lsa.length; h++)
68         temp.add(lsa[h]);
69     temp.add(attIndex, -2);
70
71 // System.out.println(r + " " + Arrays.toString(lsa));
72 // System.out.println(temp.toString());
73
74     for(at=0; at < numAt-1 && not_this_rule == false; at++)
75     {
76         a_rule = temp.get(at);
77         a_inst = (int)inst.value(at);
78
79         if( a_rule >= 0 && a_rule != a_inst)
80             not_this_rule = true;
81     }
82
83     if(not_this_rule == false)
84     {
85         int valIdxPred = ((ItemSet)m_allTheRules[1].get(r)).items()
86             ↪ [0];
87         data.instance(idx).setValue(attIndex, (double)valIdxPred);
88         if(DEBUG) System.out.println("Found " + r + " in instance " +
89             ↪ idx + "--> " + valIdxPred + ": " + inst.attribute(
90             ↪ attIndex).value(valIdxPred));
91         if(DEBUG) System.out.println(data.instance(idx).toString());
92         numFilled++;
93         done = true;
94     }
95     }
96
97     System.out.println("Filled " + numFilled + " missing values for
98     ↪ attribute " + data.attribute(attIndex).name()+".");
99
100     return data;
101 }
102
103 public static List<Integer> getAttrMissing(Instances data)
104 {
105     /*List contains attributes with missing values*/
106     List<Integer> attMis = new ArrayList<Integer>();
107     Map<Integer, Integer> attMisNum = new HashMap<Integer, Integer>();
108
109     attMis.clear();
110     AttributeStats ass = new AttributeStats();
111
112     int numAt = data.numAttributes();
113
114     System.out.println("Found missing values: ");

```

```

112
113     /* Find every attribute with MV*/
114     for(int i=0; i< numAt; i++)
115     {
116         ass = data.attributeStats(i);
117         if(ass.missingCount>0)
118         {
119             attMisNum.put(i, ass.missingCount);
120             System.out.println(data.attribute(i).name() + ": " + ass.
                ↪ missingCount);
121         }
122     }
123
124     /*Sort*/
125     // http://java2novice.com/java-interview-programs/sort-a-map-by-value/#
        ↪ sthash.WKW7RV0t.dpuf
126     Set<Entry<Integer, Integer>> set = attMisNum.entrySet();
127     List<Entry<Integer, Integer>> list = new ArrayList<Entry<Integer,
        ↪ Integer>>(set);
128     Collections.sort( list, new Comparator<Map.Entry<Integer, Integer>>()
129     {
130         public int compare( Map.Entry<Integer, Integer> o1, Map.Entry<Integer
        ↪ , Integer> o2 )
131         {
132             return (o2.getValue()).compareTo( o1.getValue() ); /*
                ↪ descending*/
133             // return (o1.getValue()).compareTo( o2.getValue() ); /*ascending
                ↪ */
134         }
135     } );
136     for(Entry<Integer, Integer> entry:list){
137         if(DEBUG) System.out.println(entry.getKey()+" ==== "+entry.getValue()
        ↪ );
138         attMis.add(entry.getKey());
139     }
140
141     return attMis;
142 }
143
144 public static Map<Integer, List<Integer>> getInstMV (Instances data, List<
        ↪ Integer> attMis)
145 {
146     /*Dictionary contains instances with missing values*/
147     Map<Integer, List<Integer>> instMis = new HashMap<Integer, List<Integer
        ↪ >>();
148     int numIns = data.numInstances();
149
150     int att = 0;
151
152     /* Find every attribute with MV*/
153     for(Iterator<Integer> i = attMis.iterator(); i.hasNext(); )

```

```

154     {
155         List<Integer> tempLst = new ArrayList<Integer>();
156         tempLst.clear();
157         att = i.next();
158
159         for (int inst = 0; inst < numIns; inst++)
160         {
161             if(data.get(inst).isMissing(att))
162                 tempLst.add(inst);
163         }
164         instMis.put(att, tempLst);
165     }
166
167     return instMis;
168 }
169
170 public static void main(String[] args) throws Exception {
171     Instances data = train_smo.load_csv (args);
172     int numAt = data.numAttributes();
173
174     /* The last index contains the class*/
175     data.setClassIndex(numAt - 1);
176
177     /*List contains attributes with missing values*/
178     List<Integer> attMis = getAttrMissing(data);
179
180     /*Dictionary contains instances with missing values*/
181     Map<Integer, List<Integer>> instMisMap = getInstMV (data, attMis);
182
183     System.out.println("Attributes containing at least a missing value: " +
184         ↪ attMis.toString());
185     System.out.println(instMisMap.get(1).toString());
186
187     Discretize discretize = new Discretize();
188     String [] options = weka.core.Utils.splitOptions("-R first-last -
189         ↪ precision 6");
190     discretize.setOptions(options);
191     discretize.setInputFormat(data);
192     data = Filter.useFilter(data, discretize);
193
194     for(int i : attMis)
195         data = fillMV (data, i, instMisMap.get(i));
196
197     /*Do it again for continents after a country has been assigned*/
198     data = fillMV (data, 4, instMisMap.get(4));
199
200     CSVSaver csv = new CSVSaver();
201     csv.setInstances(data);
202     csv.setFile(new File("newdata2.csv"));
203     csv.writeBatch();
204 }

```


B

Presupuesto

1) Ejecución Material	
▪ Compra de ordenador personal (Software incluido)	1.000 €
▪ Material de oficina	150 €
▪ Total de ejecución material	1.150 €
2) Gastos generales	
▪ sobre Ejecución Material	150 €
3) Beneficio Industrial	
▪ sobre Ejecución Material	69 €
4) Honorarios Proyecto	
▪ 2000 horas a 15 €/ hora	30000 €
5) Material fungible	
▪ Gastos de impresión	280 €
▪ Encuadernación	200 €
6) Subtotal del presupuesto	
▪ Subtotal Presupuesto	31.369 €
7) I.V.A. aplicable	
▪ 21 % Subtotal Presupuesto	6.587,49 €
8) Total presupuesto	
▪ Total Presupuesto	37.956,49 €

Madrid, Noviembre de 2015
El Ingeniero Jefe de Proyecto

Fdo.: Sergio Izquierdo del Álamo
Ingeniero de Telecomunicación



Pliego de condiciones

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un modelo de *Predicción de fraude en un Operador Móvil Virtual mediante integración de datos y data mining*. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales.

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.
2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.
3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.
4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.
5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.
7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.
8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.
9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.
10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.
11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.
12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.
13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.
14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.
15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.
16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.
18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.
19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.
20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.
21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.
22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.
23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrataz anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares.

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.
2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.
3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.
5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.
6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.
7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.
8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.
9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.
10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.
11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.
12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.