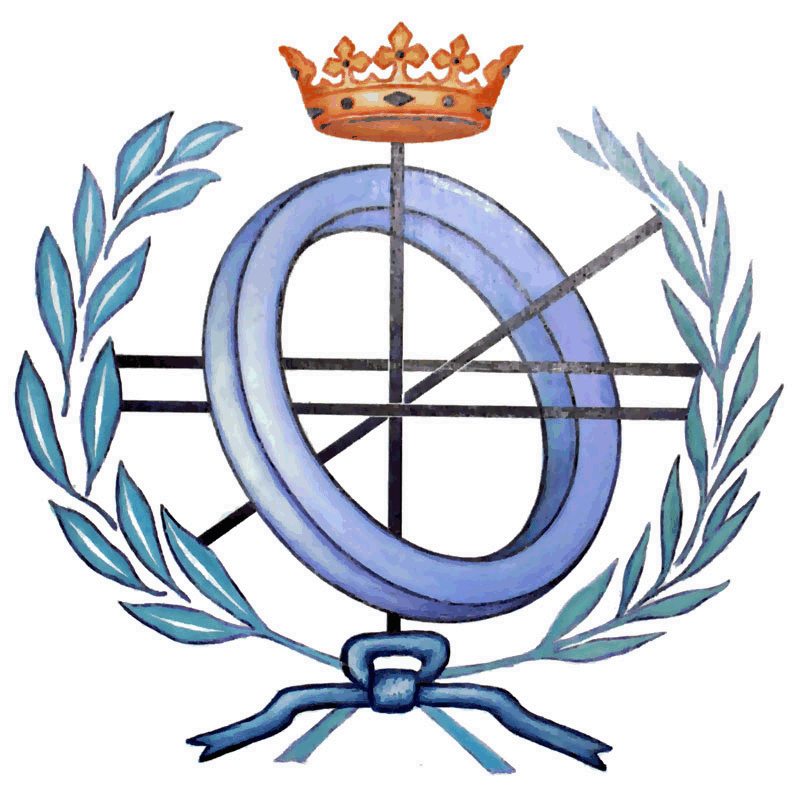
** **

**UNIVERSIDAD DE CÓRDOBA**

**ESCUELA POLITÉCNICA SUPERIOR**

**INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS**

PROYECTO FIN DE CARRERA

**Predata: librería Java para el preprocesado de datasets**

Manual Técnico

**AUTOR:** Mario Orozco Borrego

**DIRECTORES:** Prof. Dr. José Raúl Romero Salguero

D. José María Luna Ariza

**10 de septiembre de 2011**

**Prof. Dr. José Raúl Romero Salguero**

Profesor Colaborador de la Universidad de Córdoba

**D. José María Luna Ariza**

Ingeniero en Informática por la Universidad de Córdoba

**Informan:**

Que el presente Proyecto fin de carrera titulado “Predata: librería Java para el preprocesado de datasets”, que constituye la memoria presentada por Mario Orozco Borrego para aspirar al título de Ingeniería Técnica en Informática de Sistemas, ha sido realizado bajo nuestra dirección en la Escuela Politécnica Superior de la Universidad de Córdoba reuniendo, a nuestro juicio, las condiciones necesarias exigidas en este tipo de trabajos.

Y para que así conste, se expide y firma el presente certificado en Córdoba, a 10 de Septiembre de 2011.

Fdo. Fdo.

José Raúl Romero Salguero José María Luna Ariza

**D. Mario Orozco Borrego**

Alumno de la Universidad de Córdoba

**Informa:**

Que el presente Proyecto fin de carrera titulado “Predata: librería Java para el preprocesado de datasets”, ha sido realizado por él, para aspirar al título en Ingeniería Técnica en Informática de Sistemas. Todo el contenido publicado en el presente documento es original, y se presenta bajo la responsabilidad del autor.

Y para que así conste, se firma en Córdoba, a 10 de septiembre de 2011.

Fdo.

Mario Orozco Borrego

Alumno de la Universidad de Córdoba

**Tabla de Contenidos**

PARTE I. INTRODUCCIÓN AL PROYECTO 1

1. INTRODUCCIÓN 3

2. DEFINICIÓN DEL PROBLEMA 7

3. OBJETIVOS 19

4. ANTECENDENTES 21

5. RESTRICCIONES 31

6. RECURSOS 35

PARTE II. ANÁLISIS DEL SISTEMA 39

7. ESPECIFICACIÓN DE REQUISITOS 41

8. DESCRIPCIÓN DE LA INFORMACIÓN 69

9. ANÁLISIS FUNCIONAL 77

PARTE III. DISEÑO DE LA APLICACIÓN 109

10. DISEÑO PROCEDIMENTAL 111

PARTE IV. PRUEBAS 177

11. PRUEBA DE UNIDAD 179

12. PRUEBA DE LA APLICACIÓN 199

PARTE V. CONCLUSIONES Y FUTURAS MEJORAS 215

13. CONCLUSIONES Y FUTURAS MEJORAS 217

**Índice general**

I INTRODUCCIÓN AL PROYECTO........................................................................1

1 INTRODUCCIÓN 3

2 DEFINICIÓN DEL PROBLEMA 7

2.1. Definición del problema real 7

2.2. Definición del problema técnico 8

2.2.1. Funcionamiento 8

2.2.2. Entorno 10

2.2.3. Vida esperada 11

2.2.4. Ciclo de mantenimiento 12

2.2.5. Competencia 13

2.2.6. Aspecto externo 13

2.2.7. Estandarización 14

2.2.8. Calidad y fiabilidad 15

2.2.9. Programa de tareas 15

2.2.10. Pruebas 17

2.2.11. Seguridad 17

3 OBJETIVOS 19

4 ANTECENDENTES 21

4.1. Herramientas de preprocesado 22

4.1.1. Weka 22

4.1.2. RapidMiner 25

4.1.3. Knime 27

4.2. Librerías de preprocesado 29

5 RESTRICCIONES 31

5.1. Factores dato 31

5.2. Factores estratégicos 32

6 RECURSOS 35

6.1. Recursos humanos 35

6.2. Recursos materiales 36

6.2.1. Recursos hardware 36

6.2.2. Recursos software para el desarrollo 36

6.2.3. Recursos software para la implantación 37

II ANÁLISIS DEL SISTEMA 39

7 ESPECIFICACIÓN DE REQUISITOS 41

7.1. Requisitos de información 41

7.2. Requisitos funcionales 42

7.2.1. Subsistema Librería Predata 45

7.2.2. Caso de uso 1: Crear *Dataset* 46

7.2.3. Caso de uso 2: Cargar *Dataset* 47

7.2.4. Caso de uso 3: Crear Columna 48

7.2.5. Caso de uso 4: Copiar *Dataset* 49

7.2.6. Caso de uso 5: Clonar *Dataset* 50

7.2.7. Caso de uso 6: Modificar Columna 51

7.2.8. Caso de uso 7: Discretizar Columna 52

7.2.9. Caso de uso 8: Añadir Columna a *Dataset* 53

7.2.10. Caso de uso 9: Eliminar Columna de *Dataset* 54

7.2.11. Caso de uso 10: Consultar Columna *Dataset* 55

7.2.12. Caso de uso 11: Intercambiar columnas *Dataset* 56

7.2.13. Caso de uso 12: Guardar *Dataset* 57

7.2.14. Caso de uso 13: Ejecutar Estrategia 58

7.2.15. Subsistema Interfaz de usuario 59

7.2.16. Caso de uso 14: Crear Proyecto 60

7.2.17. Caso de uso 15: Crear *Dataset* Nuevo 61

7.2.18. Caso de uso 16: Importar *Dataset* 62

7.2.19. Caso de uso 17: Exportar *Dataset* 63

7.2.20. Caso de uso 18: Cargar *Dataset* en Memoria 64

7.2.21. Caso de uso 19: Aplicar operación sobre columna(s) 65

7.2.22. Caso de uso 20: Escribir *Dataset* 66

7.2.23. Caso de uso 21: Aplicar estrategia sobre *Dataset* 67

7.3. Requisitos no funcionales 68

8 DESCRIPCIÓN DE LA INFORMACIÓN 69

8.1. Formato de los datos de entrada y salida 69

8.1.1. *Dataset* KEEL 69

8.1.2. *Dataset* CSV 71

8.1.3. *Dataset* Weka 72

8.1.4. *Dataset* Excel 73

8.2. Fichero definición de estrategias 73

9 ANÁLISIS FUNCIONAL 77

9.1. Descripción arquitectónica 77

9.2. Componentes del sistema 78

9.2.1. Componente Gestor de *Dataset* 79

9.2.2. Componente Gestor de Columnas 80

9.2.3. Componente Serializador 82

9.2.4. Componente Controlador de Estrategias 82

9.2.5. Componente Excepciones 83

9.2.6. Componente Gestor E/S *Dataset* 84

9.2.7. Componente Aplicación Base 85

9.2.8. Componente Gestor operaciones y estrategias 86

9.2.9. Componente Herramientas de recursos 87

9.3. Descripción del comportamiento 87

9.3.1. Diagrama de secuencia 1: Crear *Dataset* 88

9.3.2. Diagrama de secuencia 2: Cargar *Dataset* 89

9.3.3. Diagrama de secuencia 3: Crear Columna 90

9.3.4. Diagrama de secuencia 4: Copiar *Dataset* 91

9.3.5. Diagrama de secuencia 5: Clonar *Dataset* 92

9.3.6. Diagrama de secuencia 6: Modificar Columna 93

9.3.7. Diagrama de secuencia 7: Discretizar Columna 94

9.3.8. Diagrama de secuencia 8: Añadir Columna a *Dataset* 95

9.3.9. Diagrama de secuencia 9: Eliminar Columna *Dataset* 96

9.3.10. Diagrama de secuencia 10: Consultar Columna *Dataset* 97

9.3.11. Diagrama de secuencia 11: Intercambiar columnas *Dataset* 98

9.3.12. Diagrama de secuencia 12: Guardar *Dataset* 99

9.3.13. Diagrama de secuencia 13. Ejecutar estrategia 100

9.3.14. Diagrama de secuencia 14. Crear Proyecto 101

9.3.15. Diagrama de secuencia 15. Crear *Dataset* Nuevo 102

9.3.16. Diagrama de secuencia 16. Importar *Dataset* 103

9.3.17. Diagrama de secuencia 17. Exportar *Dataset* 104

9.3.18. Diagrama de secuencia 18. Cargar *Dataset* en Memoria 105

9.3.19. Diagrama de secuencia 19. Escribir *Dataset* 106

9.3.20. Diagrama de secuencia 20. Aplicar estrategia sobre *Dataset* 107

III DISEÑO DE LA APLICACIÓN 109

10 DISEÑO PROCEDIMENTAL 111

10.1. Capa *Predata* 112

10.1.1. Componente Gestor de Dataset 112

10.1.2. Componente Gestor de columnas 119

10.1.3. Componente Serializador 149

10.1.4. Componente Controlador de estrategias 159

10.1.5. Componente Excepciones 167

10.2. Capa Interfaz de usuario 169

10.2.1. Componente Gestor E/S *Dataset* 169

10.2.2. Componente Gestor Aplicación Base 171

10.2.3. Componente Gestor Operaciones y Estrategias 171

10.2.4. Componente Herramientas de recursos 174

IV PRUEBAS 177

11 PRUEBA DE UNIDAD 179

11.1. Pruebas de los casos de uso 180

11.1.1. Caso de prueba Crear *Dataset* 180

11.1.2. Caso de prueba Cargar *Dataset* 181

11.1.3. Caso de prueba Crear Columna 181

11.1.4. Caso de prueba Copiar *Dataset* 181

11.1.5. Caso de prueba Clonar *Dataset* 182

11.1.6. Caso de prueba Modificar columna 182

11.1.7. Caso de prueba Discretizar columna 182

11.1.8. Caso de prueba Añadir columna a *Dataset* 183

11.1.9. Caso de prueba Eliminar columna de *Dataset* 183

11.1.10. Caso de prueba Consultar columna *Dataset* 183

11.1.11. Caso de prueba Intercambiar Columnas *Dataset* 184

11.1.12. Caso de prueba Guardar *dataset* 184

11.1.13. Caso de prueba Ejecutar estrategia 184

11.1.14. Caso de prueba Crear proyecto 185

11.1.15. Caso de prueba Crear *Dataset* Nuevo 185

11.1.16. Caso de prueba Importar *Dataset* 185

11.1.17. Caso de prueba Exportar *Dataset* 186

11.1.18. Caso de prueba Cargar *Dataset* en Memoria 186

11.1.19. Caso de prueba Aplicar operación sobre columna 186

11.1.20. Caso de prueba Escribir *Dataset* 187

11.1.21. Aplicar estrategia sobre *Dataset* 187

11.2. Pruebas de la librería 187

11.2.1. Caso de prueba Crear *Dataset* 188

11.2.2. Caso de prueba Cargar *Dataset* 188

11.2.3. Caso de prueba Crear columna 188

11.2.4. Caso de prueba Copiar *Dataset* 189

11.2.5. Caso de prueba Clonar *Dataset* 189

11.2.6. Caso de prueba Modificar columna 190

11.2.7. Caso de prueba Discretizar columna 190

11.2.8. Caso de prueba Añadir columna a *Dataset* 191

11.2.9. Caso de prueba Eliminar columna de *Dataset* 191

11.2.10. Caso de prueba Consultar columna de *Dataset* 192

11.2.11. Caso de prueba Intercambiar columnas *Dataset* 192

11.2.12. Caso de prueba Guardar *dataset* 193

11.2.13. Caso de prueba Ejecutar estrategia 194

11.2.14. Caso de prueba Crear proyecto 194

11.2.15. Caso de prueba Crear *Dataset* Nuevo 194

11.2.16. Caso de prueba Importar *Dataset* 195

11.2.17. Caso de prueba Exportar *Dataset* 195

11.2.18. Caso de prueba Cargar *Dataset* en Memoria 196

11.2.19. Caso de prueba Aplicar operación sobre columna 197

11.2.20. Caso de prueba Escribir *Dataset* 197

11.2.21. Caso de prueba Aplicar estrategia sobre *Dataset* 198

12 PRUEBA DE LA APLICACIÓN 199

12.1. Proceso de pruebas 199

12.1.1. Ejecución de los casos de prueba de la librería 199

12.1.2. Caso de prueba Crear *Dataset* 200

12.1.3. Caso de prueba Cargar *Dataset* 200

12.1.4. Caso de prueba Crear columna 201

12.1.5. Caso de prueba Copiar *Dataset* 201

12.1.6. Caso de prueba Clonar *Dataset* 202

12.1.7. Caso de prueba Modificar columna 202

12.1.8. Caso de prueba Discretizar columna 203

12.1.9. Caso de prueba Añadir columna a *Dataset* 203

12.1.10. Caso de prueba Eliminar columna de *Dataset* 204

12.1.11. Caso de prueba Consultar columna de *Dataset* 204

12.1.12. Caso de prueba Intercambiar columnas *Dataset* 204

12.1.13. Caso de prueba Guardar *dataset* 205

12.1.14. Caso de prueba Ejecutar estrategia 206

12.1.15. Caso de prueba Crear proyecto 206

12.1.16. Caso de prueba Crear *Dataset* Nuevo 206

12.1.17. Caso de prueba Importar *Dataset* 207

12.1.18. Caso de prueba Exportar *Dataset* 207

12.1.19. Caso de prueba Cargar *Dataset* en Memoria 208

12.1.20. Caso de prueba Aplicar operación sobre columna 209

12.1.21. Caso de prueba Escribir *Dataset* 209

12.1.22. Caso de prueba Aplicar estrategia sobre *Dataset* 210

12.2. Resultado de las pruebas 210

12.3. Solución de los problemas detectados 211

12.3.1. Solución del Caso 11 211

12.3.2. Solución del Caso 12 212

12.3.3. Solución del Caso 16 213

12.3.4. Solución del caso 36 213

12.3.5. Solución del caso 42 214

V CONCLUSIONES Y FUTURAS MEJORAS 215

13 CONCLUSIONES Y FUTURAS MEJORAS 217

13.1. Conclusiones sobre los objetivos planteados 217

13.2. Conclusión personal del autor 218

13.3. Futuras mejoras 219

BIBLIOGRAFÍA 221

**Lista de figuras**

Figura 7.1: Subsistema Librería Predata 45

Figura 7.2: Subsistema Interfaz de usuario 59

Figura 8.1: Ejemplo de un *dataset* cargado en Microsoft Excel 73

Figura 9.1: Arquitectura del sistema 78

Figura 9.2: Componente Gestor de *Dataset* 79

Figura 9.3: Componente Gestor de Columnas 80

Figura 9.4: Componente Serializador 82

Figura 9.5: Componente Controlador de Estrategias 83

Figura 9.6: Componente Excepciones 83

Figura 9.7: Componente Gestor E/S *Dataset* 84

Figura 9.8: Componente Aplicación Base 85

Figura 9.9: Componente Gestor operaciones y estrategias 86

Figura 9.10: Componente Herramientas de recursos 87

Figura 9.11: Diagrama de secuencia 1: Crear *Dataset* 88

Figura 9.12: Diagrama de secuencia 2: Cargar *dataset* 89

Figura 9.13: Diagrama de secuencia 3: Crear Columna 90

Figura 9.14: Diagrama de secuencia 4: Copiar *Dataset* 91

Figura 9.15: Diagrama de secuencia 5: Clonar *Dataset* 92

Figura 9.16: Diagrama de secuencia 6: Modificar Columna 93

Figura 9.17: Diagrama de secuencia 7: Discretizar Columna 94

Figura 9.18: Diagrama de secuencia 8: Añadir Columna a *Dataset* 95

Figura 9.19: Diagrama de secuencia 9: Eliminar Columna D*ataset* 96

Figura 9.20: Diagrama de secuencia 10: Consultar Columna *Dataset* 97

Figura 9.21: Diagrama de secuencia 11: Intercambiar columnas *Dataset* 98

Figura 9.22: Diagrama de secuencia 12. Guardar *Dataset* 99

Figura 9.23: Diagrama de secuencia 13. Ejecutar estrategia 100

Figura 9.24: Diagrama de secuencia 14. Crear Proyecto 101

Figura 9.25: Diagrama de secuencia 15. Crear *Dataset* Nuevo 102

Figura 9.26: Diagrama de secuencia 16. Importar *Dataset* 103

Figura 9.27: Diagrama de secuencia 17. Exportar *Dataset* 104

Figura 9.28: Diagrama de secuencia 18. Cargar *Dataset* en Memoria 105

Figura 9.29: Diagrama de secuencia 19. Escribir *Dataset* 106

Figura 9.30: Diagrama de secuencia 20. Aplicar estrategia sobre *Dataset* 107

Figura 10.1: Componente Gestor de Dataset 112

Figura 10.2: Clase *Dataset* 117

Figura 10.3: Clase *IIterator* 118

Figura 10.4: Clase *InstanceIterator* 119

Figura 10.5: Enumeración *ColumnType* 119

Figura 10.6: Componente Gestor de columnas 120

Figura 10.7: Clase *ColumnAbstraction* 123

Figura 10.8: Clase *BinaryColumn* 124

Figura 10.9: Clase *RangeColumn* 125

Figura 10.10: Clase *CategoricalColumn* 127

Figura 10.11: Clase *NominalColumn* 128

Figura 10.12: Clase *NumericalColumn* 131

Figura 10.13: Clase *IntegerColumn* 132

Figura 10.14: Clase *ColumnImpl* 134

Figura 10.15: Clase *BinaryColumnImpl* 136

Figura 10.16: Clase *RangeColumnImpl* 139

Figura 10.17: Clase *CategoricalColumnImpl* 142

Figura 10.18: Clase *NominalColumnImpl* 144

Figura 10.19: Clase *NumericalColumnImpl* 147

Figura 10.20: Clase *IntegerColumnImpl* 149

Figura 10.21: Componente Serializador 149

Figura 10.22: Clase *FileDataset* 150

Figura 10.23: Clase *KeelDataSet* 152

Figura 10.24: Clase *ArffDataSet* 154

Figura 10.25: Clase CsvDataSet 156

Figura 10.26: Clase *ExcelDataSet* 159

Figura 10.27: Componente Controlador de Estrategias 159

Figura 10.28: Clase *DatasetStrategy* 161

Figura 10.29: Clase *MDLPDiscretize* 162

Figura 10.30: Clase *KFolds* 163

Figura 10.31: Clase *RemoveDuplicates* 165

Figura 10.32: Clase *RemovePercentage* 166

Figura 10.33: Componente Excepciones 167

Figura 10.34: Componente Gestor E/S *Dataset* 169

Figura 10.35: Clase *LoadDatasetHandler* 170

Figura 10.36: Clase *SaveDatasetHandler* 170

Figura 10.37: Componente Gestor Operaciones y estrategias 171

Figura 10.38: Clase *FillBaseComposite* 173

Figura 10.39: Clase *FillCompleteComposite* 173

Figura 10.40: Componente Herramientas de recursos 174

Figura 10.41: Clase *ClassLoaderResourcesUtilities* 175

Figura 10.42: Clase *XMLUtilities* 175

**Lista de tablas**

Tabla 7.1: Plantilla para definir casos de uso 44

Tabla 7.2: Caso de uso 1: Crear *Dataset* 46

Tabla 7.3: Caso de uso 2: Cargar *Dataset* 47

Tabla 7.4: Caso de uso 3: Crear Columna 48

Tabla 7.5: Caso de uso 4: Copiar *Dataset* 49

Tabla 7.6: Caso de uso 5: Clonar *Dataset* 50

Tabla 7.7: Caso de uso 6: Modificar Columna 51

Tabla 7.8: Caso de uso 7: Discretizar Columna 52

Tabla 7.9: Caso de uso 8: Añadir Columna a *Dataset* 53

Tabla 7.10: Caso de uso 9: Eliminar Columna de *Dataset* 54

Tabla 7.11: Caso de uso 10: Consultar Columna *Dataset* 55

Tabla 7.12: Caso de uso 11: Intercambiar columnas *Dataset* 56

Tabla 7.13: Caso de uso 12: Guardar *Dataset* 57

Tabla 7.14: Caso de uso 13: Ejecutar Estrategia 58

Tabla 7.15: Caso de uso 14: Crear Proyecto 60

Tabla 7.16: Caso de uso 15: Crear *Dataset* Nuevo 61

Tabla 7.17: Caso de uso 16: Importar *Dataset* 62

Tabla 7.18: Caso de uso 17: Exportar *Dataset* 63

Tabla 7.19: Caso de uso 18: Cargar *Dataset* en Memoria 64

Tabla 7.20: Caso de uso 19: Aplicar operación sobre columna(s) 65

Tabla 7.21: Caso de uso 20: Escribir *Dataset* 66

Tabla 7.22: Caso de uso 21: Aplicar estrategia sobre *Dataset* 67

Tabla 11.1: Caso de prueba Crear *Dataset* 188

Tabla 11.2: Caso de prueba Cargar *Dataset* 188

Tabla 11.3: Caso de prueba Crear columna 188

Tabla 11.4: Caso de prueba Copiar *Dataset* 189

Tabla 11.5: Caso de prueba Clonar *Dataset* 189

Tabla 11.6: Caso de prueba Modificar columna 190

Tabla 11.7: Caso de prueba Discretizar columna 190

Tabla 11.8: Caso de prueba Añadir columna a *Dataset* 191

Tabla 11.9: Caso de prueba Eliminar columna de *Dataset* 191

Tabla 11.10: Caso de prueba Consultar columna de *Dataset* 192

Tabla 11.11: Caso de prueba Intercambiar columnas *Dataset* 192

Tabla 11.12: Caso de prueba Guardar *dataset* 193

Tabla 11.13: Caso de prueba Ejecutar estrategia 194

Tabla 11.14: Caso de prueba Crear proyecto 194

Tabla 11.15: Caso de prueba Crear *Dataset* Nuevo 194

Tabla 11.16: Caso de prueba Importar *Dataset* 195

Tabla 11.17: Caso de prueba Exportar *Dataset* 195

Tabla 11.18: Caso de prueba Cargar *Dataset* 196

Tabla 11.19: Caso de prueba Aplicar operación sobre columna 197

Tabla 11.20: Caso de prueba Guardar *Dataset* 197

Tabla 11.21: Caso de prueba Aplicar estrategia sobre *Dataset* 198

Tabla 12.1: Caso de prueba Crear *Dataset* 200

Tabla 12.2: Caso de prueba Cargar *Dataset* 200

Tabla 12.3: Caso de prueba Crear columna 201

Tabla 12.4: Caso de prueba Copiar *Dataset* 201

Tabla 12.5: Caso de prueba Clonar *Dataset* 202

Tabla 12.6 Caso de prueba Modificar columna 202

Tabla 12.7: Caso de prueba Discretizar columna 203

Tabla 12.8: Caso de prueba Añadir columna a *Dataset* 203

Tabla 12.9: Caso de prueba Eliminar columna de *Dataset* 204

Tabla 12.10: Caso de prueba Consultar columna de *Dataset* 204

Tabla 12.11: Caso de prueba Intercambiar columnas *Dataset* 204

Tabla 12.12: Caso de prueba Guardar *dataset* 205

Tabla 12.13: Caso de prueba Ejecutar estrategia 206

Tabla 12.14: Caso de prueba Crear proyecto 206

Tabla 12.15: Caso de prueba Crear *Dataset* Nuevo 206

Tabla 12.16: Caso de prueba Importar *Dataset* 207

Tabla 12.17: Caso de prueba Exportar *Dataset* 207

Tabla 12.18: Caso de prueba Cargar *Dataset* 208

Tabla 12.19: Caso de prueba Aplicar operación sobre columna 209

Tabla 12.20: Caso de prueba Guardar *Dataset* 209

Tabla 12.21: Caso de prueba Aplicar estrategia sobre *Dataset* 210

Tabla 12.22: Resultado de la corrección del caso 11 211

Tabla 12.23: Resultado de la corrección del caso 12 212

Tabla 12.24: Resultado de la corrección del caso 16 213

Tabla 12.25: Resultado de la corrección del caso 36 213

Tabla 12.26: Resultado de la corrección del caso 42 214

Parte I

INTRODUCCIÓN AL PROYECTO

# INTRODUCCIÓN

Hoy en día vivimos en la sociedad de la información, en la que, las tecnologías que facilitan la creación, distribución y manipulación de los datos, juegan un papel importante en las actividades sociales, culturales y económicas. Rodeados de información, la mayor parte está en espera de ser descubierta, la cual adquiere valor por la forma en la que nos relacionamos con ella y por el manejo que hacemos de la misma. En la actualidad, la información juega un papel fundamental para cualquier organización. Quienes se encargan de la toma de decisiones, han tomado cuenta de que la información no es sólo el resultado de la conducción empresarial, sino que a su vez alimenta los negocios (es una relación restrospectiva), y puede ser un factor importante a la hora de determinar el éxito o el fracaso de los mismos.

Los Sistemas y Tecnologías de Información han cambiado la visión que tienen las organizaciones respecto a la forma de operar. Mediante su utilización se logran mejoras sustanciales, computarizando procesos operativos, suministrando una plataforma de información útil para la toma de decisiones, y más importante aún, su utilización otorga ventajas competitivas o la reducción de la ventaja de la competencia.

Algunos dominios tales como comerciales, empresariales o bancarios, requieren el almacenamiento de grandes cantidades de información, que generada a partir del tratamiento automático, da lugar a lo que se conoce como conjunto de datos (más conocido en la comunidad de Minería de Datos por su anglicismo, *dataset*[1]).

Un *dataset* puede definirse como un gran conjunto de información útil almacenada, por lo general, de una manera tabular. Cada columna suele almacenar la información referente a una característica dentro del dominio determinado, el conjunto de características referidas a un solo ítem se denomina instancia. Un *dataset* puede almacenar, además, metainformación que define su propia estructura y propiedades, como el número y el tipo de atributos además de varios indicadores estadísticos que podrían ser aplicados a los mismos. En el caso más simple, un *dataset* estaría compuesto por una sola característica, consistiendo en una única columna de valores. Por lo general el orden de los datos no importa. Los valores pueden ser de los siguientes tipos.

* Numéricos reales, representan valores de tipo numérico con decimal.
* Numéricos enteros, representan valores de tipo numérico sin decimal.
* Categóricos, representan un conjunto discreto de valores [2,3].
* Nominal, representan valores que no tienen relación entre ellos.
* Binarios, representan datos que solamente pueden tomar dos valores concretos.
* Intervalares, representan datos en un rango numérico concreto.

En el caso de existir valores no conocidos, provoca la necesidad de manipular adecuadamente tanto los valores faltantes (*missing-values),* valores vacíos *(empty-values)* como los valores nulos (*null-values)*.

El principal problema existente es qué herramientas o de qué forma se debe tratar la información. En la actualidad, existen diversas herramientas para el preprocesado de *datasets*, las cuales proporcionan un entorno de trabajo donde se ofrecen las distintas operaciones que se pueden realizar. Sin embargo, esto provoca que el usuario se tenga que adaptar a las funciones concretas del sistema, negando la posibilidad de cambio alguno sobre la estructura ya preestablecida por el mismo, es decir, se trata de herramientas que no pueden utilizarse en el desarrollo de nuevo software, por lo que no es ayuda para el usuario, sino tampoco para el desarrollador. Se pretende, por tanto, invertir la situación, eliminar la barrera entre las herramientas y aquel que las use, esto es, el desarrollo de una librería que permita la gestión eficiente de *datasets* así como la inclusión de operaciones de preprocesado sobre los datos.

El preprocesado de los datos, normalmente representa hasta el 60% del esfuerzo en la extracción de conocimiento, ya que las herramientas no terminan de ser lo suficientemente flexibles como para agilizar el proceso, por lo que no resultan suficientes si lo que se quiere es desarrollar un programa propio que trabaje con datos.

En el desarrollo de este tipo de programas habría que invertir la mayor parte del tiempo en el desarrollo de un sistema que realice la gestión de los *dataset* desde el punto de vista lógico, por lo que se pretende el desarrollo de un sistema que facilite esta tarea al programador.

# DEFINICIÓN DEL PROBLEMA

En el presente capítulo, se intentará dar una idea del problema desde una visión externa, abstrayendo únicamente la problemática y las necesidades de cara al usuario final o al desarrollador que vaya a hacer uso de la librería, y posteriormente desde un punto de vista técnico, sin llegar a entrar en la solución.

## Definición del problema real

Mediante la realización del presente Proyecto se pretende el desarrollo de una librería para la gestión de *datasets*, así como la incorporación de operaciones de preprocesado. Se posibilitará la tarea al desarrollador Java de utilizar la librería en sus proyectos, así como la posibilidad de ampliar las múltiples funcionalidades que ofrece la misma, como agregar el soporte a más formatos de *dataset,* agregar nuevos tipos de columnas y ampliar la oferta inicial de algoritmos.

* El desarrollador podrá manejar el contenido del *dataset*, independientemente del formato de origen del mismo, con total libertad.
* El desarrollador podrá exportar el *dataset* en cualquier otro formato independientemente del formato origen del mismo.
* El desarrollador deberá poder modificar el contenido del *dataset* una vez cargado.
* El desarrollador será capaz de aplicar algoritmos de discretización[2,3] como MDLP[4], por igual anchura (*equal-width),* por igual frecuencia (*equal-frequency),* por tamaño de partición (*by size)* o por amplitud de partición *(by amplitude).*
* El desarrollador será capaz de detectar instancias repetidas en el *dataset* y eliminarlas.
* El desarrollador será capaz de aplicar algoritmos de particionado como KFolds[9].

## Definición del problema técnico

A continuación se llevará a cabo la identificación del problema desde el punto de vista técnico, mediante la realización de su especificación respondiendo a la PDS (*Product Design Specification).*

La PDS propone la realización de un análisis de los principales condicionantes técnicos del problema a resolver, mediante la respuesta a una serie de cuestiones básicas.

Estas cuestiones son resueltas en los diferentes subapartados que aparecen a continuación.

### Funcionamiento

En esta sección se analizarán las principales prestaciones que debe reunir el producto software que se pretende implementar mediante el desarrollo del presente Proyecto, expresando de esta forma en lenguaje técnico aquellas características del problema real expuesto con anterioridad que se consideren relevantes a la hora de abordar el proceso de creación del mismo.

* Se permitirá la lectura y escritura de cuatro tipos de *dataset*: KEEL[5], Weka[6], CSV (*Comma Separated Values)*[7] y Excel[8].
* Se definirán los siguientes tipos de columna:
  + Columna Numérica Real: representa valores numéricos con decimal.
  + Columna Numérica Entera: representa valores numéricos sin decimal.
  + Columna Categórica: representa un conjunto discreto de valores.
  + Columna Binaria: este tipo de columna representa valores de tipo binario, es decir, solamente puede tomar dos valores determinados.
  + Columna Nominal: almacena valores que no tienen relación entre ellos.
  + Columna de Rango: cada valor representa un intervalo numérico acotado.
* El contenido del *dataset*, esto es, los datos en sí, se almacenarán en la estructura del *dataset*, ordenados por columnas, es decir, se podrá realizar un acceso al contenido del *dataset* indicando la columna en un primer lugar y posteriormente al elemento en concreto de la misma
* Se permitirá el manejo del contenido de las columnas, así como del contenido del *dataset,* esto es, añadir o eliminar columnas a conveniencia.
* Se permitirá el recorrido del contenido del dataset por instancias.
* La librería incorporará las funciones necesarias para permitir el manejo de valores nulos (*null-values),* valores vacíos (*empty-values)* o valores faltantes (*missing-values)*.
* Se diferenciarán las operaciones de preprocesado en algoritmos aplicables sobre un *dataset* completo y algoritmos aplicables sobre atributos (columnas).
* Se implementarán cuatro algoritmos aplicables sobre *datasets*, discretización por entropía, eliminación de instancias repetidas, eliminación de porcentajes de instancias y partición de *datasets* siguiendo el método KFolds[9].
* Se implementarán discretizaciones sobre columnas, mediante los siguientes métodos.
  + Discretización por igual anchura, se realiza un particionado del conjunto continuo de los datos, de forma que especificando un número de particiones, cada una de ellas tenga la misma amplitud.
  + Discretización por igual frecuencia, el particionado se realiza de forma que indicando el número de particiones, en cada una de ellas, la frecuencia de aparición de los datos sea la misma.
  + Discretización por igual tamaño de particiones, partiendo de un tamaño determinado de partición, se calcula el número de particiones necesarias para conseguir que todas tengan el tamaño especificado.
  + Discretización por igual amplitud de particiones, especificando la amplitud de partición, se computa el número de particiones necesarias para que cada partición tenga la misma amplitud, es decir, la diferencia entre el valor máximo y el mínimo es igual al tamaño de partición especificado.

### Entorno

El entorno se define como el conjunto de aspectos o propiedades que rodean al problema, y que, aún presentándose de manera externa al mismo, pueden influir en el planteamiento de una solución, puesto que pueden afectar al sistema software desarrollado para tal fin.

Este término resulta demasiado amplio, por lo que para definirlo se tendrán en cuenta los siguientes puntos: entorno de programación, entorno software, entorno hardware y entorno de usuario.

**Entorno de programación:** este software se va a desarrollar en el lenguaje de programación Java. Como entorno de programación se usará la plataforma de desarrollo Eclipse[10] versión 3.6.2. Esta aplicación proporciona una interfaz de desarrollo adecuado para el problema que se abordará, además brinda un entorno de depuración de código excelente.

**Entorno software:** para la correcta ejecución de este software es necesario disponer de la Máquina Virtual Java[11]. La naturaleza multiplataforma que presenta la Máquina Virtual de Java, determina la posibilidad de utilizar diferentes sistemas operativos, cuya elección queda relegada por tanto a un segundo plano. Existen implementaciones de máquinas virtuales disponibles para un gran número de entornos, entre los cuales podría incluirse cualquier versión estable de los sistemas operativos Linux o Unix, Microsoft Windows 9x/Me/2000/XP/Vista/7, o Macintosh OS 9/X. La librería a desarrollar hará uso de la librería Apache POI[12], que proporciona la funcionalidad requerida para manejar documentos procedentes de entornos Microsoft Office.

**Entorno hardware:** para la ejecución y utilización de la librería, únicamente se requiere una máquina con los suficientes recursos para ejecutar la Máquina Virtual Java.

**Entorno de usuario:** la librería que se pretende desarrollar en el presente Proyecto ha sido planteada de forma que resulte sencilla en intuitiva, permitiéndose su utilización por parte de cualquier desarrollador que tenga conocimientos generales de programación en Java.

### Vida esperada

La vida esperada de un producto software se puede definir como el tiempo estimado durante el cual puede realizarse una aplicación útil del mismo. Se trata de una estimación difícil de realizar, al influir en la misma numerosos factores, a pesar de lo cual a continuación se realizará un análisis de los condicionantes que se consideran de mayor relevancia.

* El lenguaje de programación Java, en el que se desarrollará el sistema software, se encuentra en continua evolución, por lo que podría resultar interesante realizar actualizaciones periódicas de la librería a fin de incorporar las nuevas funcionalidades que surjan, incrementando de este modo el rendimiento de la misma.
* Una parte del sistema software que se pretende desarrollar hace uso de la librería Apache POI. Como consecuencia, esta parte se verá afectada en gran medida por la evolución de la librería Apache POI, por lo que sería interesante realizar el mantenimiento periódico de esta parte con el fin de incorporar las nuevas funcionalidades.

### Ciclo de mantenimiento

El ciclo de mantenimiento se identifica con el conjunto de modificaciones que resulta necesario realizar para que una determinada aplicación pueda hacer frente a diferentes circunstancias que puedan surgir, o a nuevas exigencias procedentes del usuario final o del propio sistema. Las diferentes circunstancias deberán ser analizadas y diseñadas para una corrección por parte de los programadores informáticos. Se van a nombrar los factores que se prevé que podrán ser causa de un futuro mantenimiento.

**Perfeccionamiento:** Podrá surgir la necesidad de realizar modificaciones adicionales con el objetivo de introducir mejoras en el sistema, completando la funcionalidad proporcionada por el mismo. De cara a facilitar el mantenimiento, el diseño de la librería se hará de forma muy modular, encapsulando cada función lo máximo posible, facilitando la tarea de extender su funcionalidad.

**Adaptación:** Será necesario analizar los cambios que se producen en la librería Apache POI, para evitar posibles incompatibilidades que puedan surgir con el código de la librería.

**Corrección:** El diseño modular y las clases se realizará de un modo que se facilite lo máximo el proceso de búsqueda y corrección de errores.

### Competencia

Actualmente no existe ninguna librería Java que implemente la misma funcionalidad requerida para el Proyecto. No obstante existen herramientas para la manipulación de *datasets*, pero que proporcionan un entorno restringido donde se establece una dependencia, en la que se está expuesto a las modificaciones de terceros sobre las herramientas. A continuación se mencionan las más importantes.

* Weka: herramienta que contiene una colección de algoritmos para el análisis de datos bastante restringida en cuanto a preprocesado, mediante una interfaz gráfica o en línea de comandos, escrita en Java.
* Knime[13]: proporciona distintos algoritmos para la manipulación de datos y análisis.
* RapidMiner[14]: al igual que Knime proporciona herramientas para la manipulación y análisis de datos.

### Aspecto externo

El aspecto externo de la aplicación hace referencia no solamente al impacto visual que recibe el usuario al realizar la ejecución de la misma, siendo necesario considerar la presentación física del sistema, los mecanismo de instalación proporcionados junto al mismo y los manuales que puedan acompañarlo, tanto en formato papel (Manual de Instalación, Manual de Usuario) como en formato digital (ayuda en línea). Se van a considerar los siguientes aspectos.

**Interfaz de usuario:** es necesario mencionar que el sistema a desarrollar no presentará una interfaz gráfica. El mecanismo que utilizará el usuario para la interactuación con el sistema es el propio de una librería de clases en Java. No obstante se llevará a cabo el desarrollo de una interfaz de usuario de para las pruebas de los algoritmos desarrollados en la librería.

**Formato de almacenamiento y distribución:** todo el material generado por el proyecto será entregado en formato digital en un soporte de tipo CD-ROM, debido a la gran popularidad de este formato, así como su bajo coste. Este incluirá el software necesario para permitir su correcto funcionamiento, es decir, las librerías externas requeridas para su ejecución, en este caso Apache POI, la cual se proporcionará en formato JAR. En el soporte se incluirá además de la aplicación, toda la documentación generada.

**Documentación**: la documentación incluida en el soporte que se entregará el Proyecto incluirá, el Manual Técnico, que reúne las fases de búsqueda de requisitos, análisis, diseño y prueba del sistema; el Manual de Código, que reunirá todo el proceso de implementación de la aplicación resultante; y el Manual de usuario, que explicará el uso de la aplicación de una forma sencilla.

### Estandarización

La estandarización hace referencia al conjunto de criterios que regulan ciertos aspectos relativos a las aplicaciones, a fin de permitir y facilitar la interrelación de éstas entre sí y con el usuario final de las mismas.

Este concepto abarca un gran numero de aspectos distintos, por lo que a continuación se realizará un análisis independiente de aquellos que se consideran de mayor relevancia.

**Lenguaje de programación:** el lenguaje de programación utilizado en la codificación de la librería será Java 6. La utilización de cualquier funcionalidad propia de este lenguaje, así como de las bibliotecas proporcionadas en su versión oficial, incluidas en la interfaz de programación de aplicaciones (API ≈ *Application Programming Interface*) correspondiente a esta versión, se encuentran totalmente verificadas, de forma que se garantiza la compatibilidad del código ejecutable con cualquier máquina virtual de Java actualizada, independientemente de la plataforma que se desee para la ejecución del sistema. La utilización de cualquier lenguaje de programación recomienda emplear una serie de normas relativas al estilo de codificación, respecto a esto, se utilizará la Notación Húngara [16].

**Documentación del código:** la documentación del código está sometida a un estándar, impuesto por JavaDoc [17], la cual se trata de una herramienta de generación de documentación automática, a partir del código obtiene un manual de referencia en formato HTML, cuya utilización presenta numerosas ventajas debido a la calidad de la presentación y la facilidad de navegación.

### Calidad y fiabilidad

El software será sometido a un estudio para su definición y diseño así como a un conjunto de pruebas con el fin de determinar posibles errores cometidos a lo largo del desarrollo del mismo, de forma que el producto final se encuentre libre de errores de diseño y codificación. De esta forma se asegurará que la aplicación ejecuta de forma correcta sus funciones y de la mejor forma posible.

### Programa de tareas

El programa de tareas se define como el conjunto de etapas y actividades que constituyen el proceso de desarrollo de una aplicación. A continuación, se describirán las diferentes etapas en las que se puede organizar el proceso de desarrollo de esta herramienta software.

**Fase de obtención de conocimientos:** en esta fase se estudiarán aquellas áreas relacionadas con la temática del Proyecto, o necesarias para su futuro desarrollo.

* Estudio del lenguaje de programación Java.
* Estudio de la plataforma de desarrollo Eclipse, así como de las APIs para la implementación de interfaces gráficas SWT/Jface[18], para el desarrollo de la interfaz de usuario.
* Estudio de los diferentes formatos de *datasets* disponibles.
* Estudio de la API Apache POI, para la manipulación de tipos de archivos Excel, con extensión *xls* y *xlsx*.
* Estudio de los distintos algoritmos existentes relacionados con el preprocesado de *datasets*, como discretización.

**Fase de análisis y diseño:** durante esta fase se despliegan los aspectos de diseño relacionados con operaciones complejas sobre *datasets*, algoritmos aplicables al preprocesado, haciendo hincapié en los requerimientos que deben tener cada uno de ellos, y que es la motivación de los objetivos que se plantearán para el Proyecto.

**Fase de implementación:** en esta fase se implementará todo lo previamente diseñado, haciendo uso del lenguaje de programación Java, así como de las APIs comentadas y de la interfaz de desarrollo Eclipse.

**Fase de prueba:** durante esta fase se llevarán a cabo las pruebas para las distintas funcionalidades implementadas y se observará el efecto que deriva de su aplicación.

**Fase de documentación:** se completará la memoria principal del Proyecto, resumiendo los resultados obtenidos. Por último se redactará el Manual de Usuario y de Código.

### Pruebas

Durante la fase de codificación del software, se realizarán pruebas de caja blanca. Este tipo de pruebas estarán orientadas al descubrimiento de errores de codificación que provoquen fallos en la funcionalidad interna que se desarrolle.

Tras la codificación, será necesario establecer otro tipo de pruebas que proporcionen información acerca del funcionamiento general del sistema. Estas pruebas, denominadas pruebas de caja negra, permitirán comprobar si se cumplen las especificaciones y funcionalidades que se diseñaron al comienzo del Proyecto.

### Seguridad

La seguridad que presentará la herramienta que se pretende desarrollar deberá consistir en garantizar que durante su ejecución, no se realice ninguna actividad incorrecta ajena a su propia funcionalidad. No se especifica ningún nivel de seguridad más allá del que proporciona el lenguaje de programación Java. Los datos de entrada y los resultados obtenidos serán de carácter publico. La información con la que se trabaja y los resultados generados por el sistema carecen de privacidad.

# OBJETIVOS

El objetivo de este Proyecto es el desarrollo de una librería Java escalable que permita la representación y gestión de *datasets*, así como la aplicación de operaciones complejas sobre los mismos. Deberá contar con un mecanismo simple que permitirá al desarrollador extender la funcionalidad de la misma sin requerir cambio alguno sobre los componentes previamente diseñados, o en el peor de los casos, con el mínimo cambio posible en el núcleo de la librería.

Este objetivo se descompone en los siguientes subobjetivos.

* Recopilación y estudio comparativo de técnicas de preprocesado ya implementadas en otras herramientas como Weka[6], Knime[13] o RapidMiner[14], como:
* Discretización[2,3], consiste en obtener desde un conjunto de datos un subconjunto finito de modo que tenga las mimas propiedades y características que el conjunto inicial.
* Filtrado *(Filtering)*, se trata de tomar el conjunto completo de datos, obteniendo un subconjunto del mismo acorde a unos criterios preestablecidos.
* Nominal a Binario (*Nominal to Binary)*, se transforman los valores binarios, reemplazando cada atributo nominal de k valores, por k-1 valores binarios.
* Filtro de transformación numérica (*NumericTransformFilter)*, convierte atributos numéricos usando un método dado.
* Diseño y desarrollo de un *framework* para representación de *datasets* y operaciones básicas sobre instancias y características.
* Lectura y escritura de *datasets* desde/hacia ficheros de distintos formatos (CSV[7], Weka (*arff*), Excel (*xls/xlsx*) y KEEL[5]) y tipado, donde se obtiene un *dataset* que incorpora métodos para el acceso de la información, en lugar de acceder directamente a los datos en sí.
* Representación de diferentes categorías de características (nominales, categóricos, numéricos reales, numéricos enteros, intervalares y binarios).
* Tratamiento de valores nulos, vacíos y faltantes.
* Desarrollo de algoritmos de discretización básicos y *splitting* supervisados, como MDLP[4] o KFolds[9], y discretización no supervisada (*equal width, equal frequency, by amplitude* y *by size)*.
* Desarrollo de un interfaz de usuario sobre Eclipse, para la prueba de los algoritmos implementados en la librería.

# ANTECENDENTES

En los últimos años, la Minería de Datos (*Data Mining*) ha experimentado un auge como soporte para la gestión de la información y el conocimiento. Ésta permite explorar y analizar conjuntos de datos disponibles para ayudar en la toma de decisiones. El principal problema del que adolece la Minería de Datos, es las inconsistencias existentes en estos conjuntos de datos. Para solucionarlo se hace uso del preprocesado de *datasets* o de los datos en su concepto más amplio.

El preprocesado de *datasets*[1] puede definirse como todas aquellas técnicas de análisis de datos que permiten mejorar la calidad de un conjunto de datos, de modo que, las técnicas de extracción de conocimiento o Minería de Datos puedan obtener mayor rendimiento y extraer resultados de mayor calidad.

Actualmente existen diversas aplicaciones que podrían ser clasificadas como herramientas y librerías. En la primera de ellas, se ofrece al usuario la posibilidad de hacer uso de los métodos implementados en la misma; en la segunda, es posible hacer uso de los métodos, además de ampliar su funcionalidad.

A continuación se muestra una clasificación entre herramientas y librerías que existen actualmente, y se han considerado de importancia su análisis antes del diseño del sistema software que se pretende desarrollar, para de esta forma entender mejor cuales son los puntos fuertes y débiles de estas aplicaciones.

## Herramientas de preprocesado

### Weka

Para definir cada una de las técnicas de Weka, se seguirá la misma clasificación usada en la aplicación, es decir, se dividirá el conjunto de operaciones, en técnicas supervisadas y técnicas no supervisadas.

#### No supervisadas

* *Add*: añade una columna más al *dataset.* Como parámetros se ha de especificar el índice de la posición que ocupará esta nueva columna (empezando desde el 1), el nombre del atributo y si se trata de una columna de tipo discreto (conjunto finito de valores), es decir, se especificarán los posibles valores separados por comas (en caso de que no se especifique, el atributo será numérico).
* A*ddNoise*: añade ruido a un determinado atributo que debe ser nominal, especificando el porcentaje de ruido a introducir, la semilla para el generador de números aleatorios que se usa al generar dicho ruido, y por último, si se han de tener en cuenta los valores faltantes.
* *Discretize*: discretiza un conjunto de valores numéricos para convertirlos en valores nominales (por rangos). Como parámetros toma los índices de las columnas a discretizar, el número de particiones que se desean obtener, si se desea, es posible indicar que se realice una búsqueda del número de particiones óptimas (usando una distribución de probabilidad logarítmica para el ajuste de las fronteras de cada partición *log-likelihood cross validated* [19]). Da la posibilidad de aplicar la operación según dos métodos: según la frecuencia de aparición en cada intervalo (*equal-frequency)* o por igual tamaño de las particiones (*equal-width).*
* *MakeIndicator:* crea una nueva columna binaria, a partir de la original, asignará valor verdadero en caso de que un valor de la columna se encuentre en una columna especificada como parámetro de entrada, en caso contrario asignará el valor de falso.
* *NominalToBinary:* transforma todos los valores nominales de una columna dada, en un vector cuyas coordenadas son binarias.
* *Normalize:* normaliza todos los datos de manera que el rango de los mismos pasa a ser [0,1]. Para la normalización se usa la formula:
* *NumericToBinary:* convierte datos en formato numérico a binario. Si el valor de un dato es 0 o desconocido, el valor binario resultante será 0.
* *PKIDiscretize:* realiza una discretización del espectro continuo de valores usando *equal-frequency* o *equal-width*[2,3]*.* El número de particiones resultantes es igual a la raíz cuadrada del número de valores no faltantes, es decir, de valores presentes. Fue desarrollada específicamente para Weka.
* *PrincipalComponents:* se trata de una técnica de extracción de atributos, la extracción de atributos permite la representación de la misma información pero de distinta manera, ampliando el espacio de atributos para conseguir una mayor separabilidad, o reduciéndolo para reducir la dimensionalidad.
* *RandomProjection:* reduce la dimensionalidad de los datos, proyectándola en un subespacio de menor dimensionalidad utilizando para ello una matriz aleatoria. A pesar de reducir la dimensionalidad de los datos resultantes se procura conservar la estructura y propiedades fundamentales de los mismos. Como parámetros toma el número características donde queremos aplicar la operación y el tipo de distribución de la matriz aleatoria.
* *RandomSubset:* selecciona un subconjunto aleatorio de atributos, como valor absoluto o como porcentaje, los parámetros son la semilla para la selección aleatoria de los valores (*seed*) y el número de atributos a escoger, si es menor que 1, se computa como porcentaje, si es mayor que 1 se computa como valor absoluto.
* *RELAGGS:* se trata de un algoritmo de proposicionalización, en el que se trata de transformar un problema relacional en una representación de atributo-valor que pueda ser usada por los algoritmos más convencionales de aprendizaje. Basado en el algoritmo *RELAGGS*[20], de ahí su nombre.
* R*emove:* borra un conjunto de columnas del *dataset*.
* *RemoveType*: elimina un conjunto de columnas de un tipo determinado.
* *ReplaceMissingValues*: reemplaza todos los valores faltantes, por la moda en el caso de que sea una atributo nominal o la media aritmética si es un atributo numérico.
* *StringToNominal*: convierte un atributo de tipo cadena de caracteres en un tipo nominal.
* *NonSparseToSparse*: convierte todas las instancias a un formato *sparse,* este formato, puede ser visto como un matriz en la que la mayoría de los elementos son 0, por lo que de esta forma se está simplificando la representación interna del *dataset.*
* *Randomize*: realiza un ordenado aleatorio de las instancias, recibe como parámetro la semilla para generar la aleatoriedad.
* *RemoveFrequentValues:* realiza un filtrado de valores, teniendo en cuenta su frecuencia, si se pasa como parámetro un número N de valores, se extraerán los N valores con la frecuencia más alta del conjunto.
* *RemovePercentage:* elimina un porcentaje del *dataset* pasado como parámetro.
* *RemoveWithValues:* elimina las instancias que cumplan una determinada condición pasada como parámetro.
* *Resample:* obtiene un subconjunto de instancias, partiendo del conjunto inicial, elegido de forma aleatoria, acepta como parámetros, la semilla para generar los números aleatorios y el porcentaje de la muestra original a tomar.
* *ReservoirSample:* produce un subconjunto aleatorio del *dataset*, usando el algoritmo *reservoir sampling Algorithm “R”*[20].
* *SubsetByExpression:* extrae un subconjunto de instancias según una expresión de entrada.

#### Supervisadas

* *Discretize*: discretiza un conjunto de valores numéricos para convertirlos en valores nominales (por rangos). Usando un algoritmo basado en entropía, con criterios de parada: MDL de Fayyad & Irani [4] (usado por defecto) ó de Kononeko [22].
* *SMOTE:* extrae un subconjunto del *dataset* aplicando la técnica conocida como *Synthetic Minority Oversampling Technique (SMOTE)*[23].

### RapidMiner

Para clasificar las técnicas implementadas en RapidMiner se seguirá la estructura definida en el propio programa.

#### Modificación de nombre y rol *(Name and Role Modification)*

* *Rename:* se trata de una operación para modificar el nombre de una columna, toma como parámetros, el atributo a modificar, y el nombre nuevo que se le quiere asignar.
* *Rename by replacing:* renombra una columna, pero toma como parámetros una expresión regular que será el parámetro que se modificará, y el nombre nuevo que tomará la columna.
* *Rename by generic names*: renombra columnas, de tal forma que cuyo nombre sea un nombre con un prefijo del tipo att1, att2, att3, etc. El prefijo es el parámetro de entrada.

#### Transformación de datos *(Data Transformation)*

* *Discretize by Size*: lleva a cabo una discretización del espectro continuo de los datos, especificando el tamaño de las particiones, en el caso de que no se pueda obtener el tamaño especificado, el resultado puede variar ligeramente.
* *Discretize by Binning:* se realiza una discretización especificando el número de particiones a obtener.
* *Discretize by Frequency:* realiza una discretización de los atributos numéricos del *dataset*. Se realza de forma que exista una igual frecuencia de atributos en cada partición.
* *Discretize by Entropy:* se lleva a cabo una discretización, en este caso se usa una técnica distinta, llamada de entropía. La entropía mide la incertidumbre que tiene una variable de salida ante un suceso S. Si S tiene N salidas posibles, entonces la entropía de S es:

Se selecciona como un valor de umbral (que divide el conjunto de datos continuo que se trata en cada instante) que minimiza la entropía, entonces el método es aplicado recursivamente para ambas particiones, hasta que se alcance alguna condición impuesta por un criterio de parada. Usa el método MDL de Fayyad&Irani para el modo supervisado, y para el no supervisado, el propuesto por Dougherty, Kohavi y Sahami [24].

* *Replace Missing Values:* se reemplazan los valores faltantes en el dataset, de forma que, si un valor no existe, entonces es reemplazado por el mínimo de la columna, máximo o media (a elegir).
* *Sample:* se obtiene un subconjunto aleatorio del dataset original, existen tres modos, absoluto, donde se indica el tamaño del subconjunto; relativo, donde se indica el porcentaje de la muestra inicial; y probabilístico, donde se indica la probabilidad de una extracción para un subconjunto.
* *Split Data:* realiza una partición de los datos, en subconjuntos acorde a un tamaño relativo previamente especificado.
* *Remove Duplicates:* elimina aquellas instancias duplicadas en el *dataset.*
* *Sort:* ordena el *dataset* según una dirección especificada (ascendente o descendente).
* *Sort by Pareto Rank:* se realiza un ordenamiento del *dataset*, usando el método *ParetoRank* [25].
* *Shuffle:* crea un nuevo *dataset* desordenado aleatoriamente creando una copia en memoria del mismo.

### Knime

Se seguirá la misma estructura definida en el propio programa.

#### Particionado *(Binning)*

* *Numeric Binner:* operador usado para obtener un *dataset*, donde las particiones son las indicadas como parámetros de entrada del operador. Es posible indicar para cada columna, las particiones y para cada partición el rango numérico de datos asignado a cada una de ellas.
* C*AIM Binner:* se realiza una discretización usando el algoritmo CAIM [26], que crea todas las particiones posibles y elige la que minimiza la interdependencia entre clases. Para reducir el tiempo de ejecución la implementación crea solamente aquellas particiones en las que el valor del extremo y la clase cambian.

#### Conversión y reemplazo *(Convert and Replace)*

* *Number to String:* realiza la conversión de cada uno de los valores numéricos de una columna, a cadenas de caracteres (*string).*
* *String to Number:* se lleva a cabo la conversión de cada unos de los valores de una columna de tipo cadena de caracteres a valores numéricos.
* *Double to Int:* se hace una conversión de los valores de una columna, de tipo real a tipo entero, realizando un redondeo, los disponibles son los homólogos a las funciones Java: *round, ceil o floor.*
* *Rename:* permite renombrar o cambiar el tipo de columnas seleccionada.
* *String Replace (Dictionary):* reemplaza cada celda en la columna seleccionada, cada elemento es comparado con las claves de un diccionario y se reemplaza por la correspondiente en el mismo.
* *String Replacer*: reemplaza cada elemento de una columna por el especificado como parámetro, si el elemento coincide con cierta expresión regular introducida como parámetro.

#### Separación y Unión *(Split & Combine)*

* *Cell Splitter:* realiza una división de una columna de valores de tipo nominal, se realiza la división especificando un símbolo delimitador que la dividirá.
* *Cell Splitter By Position:* divide la columna de tipo nominal seleccionada en un número determinado de columnas, es dividida según una posición indicada dentro de cada valor de la columna, es decir, se divide el valor nominal en sí, tratándolo como cadena de caracteres.
* *Column Combiner:* combina el contenido de un conjunto de columnas dentro de una nueva columna, donde se concatenarán los valores como si se trataran de cadenas de caracteres.

#### Transformación *(Transform)*

* *Case Converter*: convierte los valores alfanuméricos de cada celda a mayúscula o minúscula.
* *SMOTE*: extrae un subconjunto del *dataset* aplicando la técnica conocida como *Synthetic Minority Oversampling Technique (SMOTE)* anteriormente comentada.
* *Normalizer*: este método normaliza todos los valores numéricos de la columna seleccionada. Es posible seleccionar el método de normalización, los disponibles son: normalización lineal, normalización N(0,1) y normalización por escalado decimal.

## Librerías de preprocesado

En cuanto a librerías enfocadas al desarrollador, al inicio del análisis del problema que se está abordando en el presente Proyecto, se han considerado como más relevantes dos de ellas, Java-ML[27] y JDMP[28].

La primera de ellas es una librería por lo que no posee interfaz gráfica, que contiene una colección de algoritmos relacionados con el *data mining*. Se presenta como una librería que tiene una interfaz muy clara y común para todos los algoritmos, con código excelentemente comentado y con multitud de ejemplos y tutoriales.

Se trata más bien de una librería para la extracción de conocimiento que para preprocesado, incluye escasas operaciones de preprocesado, únicamente tiene las necesarias para la correcta ejecución de sus algoritmos, por lo que no está enfocada al preprocesado. De hecho su nombre Java-ML es la abreviatura de *Java Machine Learning Library.* Es dependiente de Weka.

En cuanto a JDMP es una librería *open source* orientada al análisis de datos y *machine learning*. Facilita en gran medida el acceso a los datos, por lo que no hay que preocuparse por aspectos técnicos a la hora de importar datos de *datasets* en formatos como *TXT, CSV, Excel, MATLAB, Latex, MTX, HTML, WAV o BMP,* la multitud de formatos que soporta es el punto fuerte de esta librería.Incluye multitud de algoritmos de aprendizaje como agrupamiento, regresión, clasificación, y optimización, en cada uno de las operaciones que incluye, incorpora un módulo gráfico para la visualización de los resultados de una forma muchísimo más clara e intuitiva. Cabe destacar que hace uso de una biblioteca con la capacidad de manejar matrices enormes, además hace uso de Weka.

# RESTRICCIONES

En este capítulo se enumerarán los distintos factores que de algún modo condicionarán el desarrollo del Proyecto hacia un camino determinado. Se enumerarán a continuación dividiéndose en dos grupos.

* Factores dato, impuestos por las características del Proyecto y por lo tanto invariantes.
* Factores estratégicos, cuyo valor ha sido establecido por los técnicos proyectistas en función de las distintas alternativas existentes.

## Factores dato

Se van a considerar los siguientes factores dato.

* Uno de los requisitos principales del Proyecto es el desarrollo del sistema software en el lenguaje de programación Java, al presentar numerosas ventajas derivadas de su naturaleza multiplataforma y orientada a objetos.
* Como consecuencia de la codificación en lenguaje Java, resultará necesario abordar el proceso de especificación de requisitos y el proceso de diseño de la aplicación desde una perspectiva orientada a objetos.
* El software utilizado para el desarrollo del Proyecto deberá ser de distribución gratuita y a ser posible de licencia libre, debido a las restricciones económicas al tratarse de un Proyecto Fin de Carrera. En caso de no resultar posible emplear software gratuito, se intentará hacer uso de los recursos que de los que disponga el autor del Proyecto, así como de aquellos que pueda proporcionar el Departamento de Informática y Análisis Numérico de la Universidad de Córdoba.
* El hardware utilizado en el desarrollo del Proyecto deberá ser el que pueda ser proporcionado por el propio autor, sin embargo, no es un problema debido a que el Proyecto no requiere un equipamiento hardware especial.

## Factores estratégicos

Se van a considerar los siguientes factores estratégicos, acompañados de su análisis y motivo de elección de los mismos.

* La librería desarrollada deberá ser modular, de manera que permita la realización de futuras modificaciones y ampliaciones que puedan considerarse necesarias, con facilidad y en el menor tiempo posible. De esta forma, se permitirá la escalabilidad, característica que se considera primordial a fin de permitir la incorporación a la librería de nuevos componentes que incrementen la funcionalidad de la aplicación desarrollada.
* La plataforma de desarrollo utilizada será Eclipse, que incorpora las características propias del lenguaje y la programación Java. Eclipse representa un entorno de implementación de aplicaciones desarrollado bajo el paradigma del software libre, con naturaleza multiplataforma y multilenguaje. Se encuentra implementado en lenguaje Java, e incluye numerosas extensiones y herramientas de desarrollo que aportan al programador un elevado número de prestaciones que facilitan el proceso de creación de aplicaciones.
* El método de diseño del sistema se hará conforme a un diseño orientado a objetos, debido a las grandes ventajas que ello supone y de cara al Proyecto que nos ocupa es especialmente beneficioso ya que el diseño se hará mucho más simple, además de favorecer las futuras modificaciones sobre el trabajo ya realizado.

# RECURSOS

En este punto se expondrán los diferentes recursos de los que dispone el Proyecto para su realización. Se hará, desde un punto de vista humano y material.

**Recursos humanos:** se refiere al conjunto de personas que intervendrán de un modo u otro en el transcurso del Proyecto.

**Recursos materiales**: son aquellos que pueden definirse como el conjunto de todas la entidades no animadas que permiten realizar el proceso de desarrollo de la aplicación, así como la generación de la documentación relativa a la misma.

## Recursos humanos

Las personas que intervendrán en el desarrollo del Proyecto se enumeran a continuación.

* Directores del Proyecto: el director del Proyecto, el Prof. Dr. José Raúl Romero Salguero, y como codirector, D. José María Luna Ariza, que serán los encargados de guiar al autor durante el desarrollo del Proyecto, iniciarle sobre el dominio y alcance del problema que se va a resolver y servir de apoyo a las dificultados que puedan surgir. Ambos se encargarán de supervisar las tareas durante el desarrollo del Proyecto para comprobar que los resultados obtenidos se adecuan a los requisitos iniciales.
* Autor del Proyecto: D. Mario Orozco Borrego, será el encargado de realizar la planificación temporal del Proyecto, diseñar el software, implementarlo y realizar las pruebas necesarias para verificar su correcto funcionamiento, así como, el cumplimiento de los requisitos inicialmente planteados. Generará la documentación del Proyecto a lo largo del desarrollo del mismo, que constará de Manual Técnico, Manual de Usuario y Manual de Código.

## Recursos materiales

El conjunto de herramientas utilizadas en el proceso de desarrollo del Proyecto pueden organizarse en dos categorías principales.

* Recursos hardware: conjunto de equipos informáticos empleados para el Proyecto.
* Recursos software: conjunto de programas software que son utilizados por el equipo de desarrollo y que se ejecutan bajo una serie de equipos hardware. Existen dos tipos de recursos software: para el desarrollo y para la implantación.

### Recursos hardware

Para el desarrollo del presente Proyecto, se usarán los siguientes recursos hardware.

* Ordenador portátil Macbook Pro 15” (Mid 2009). Procesador Intel Core2 Duo 2.66Ghz. Disco duro SATA de 320gb a 5200rpm. 4GB de memoria RAM. Tarjeta gráfica NVIDIA Geforce 9600M GT de 256MB de memoria.

### Recursos software para el desarrollo

Los recursos asociados al desarrollo y edición de la documentación serán los siguientes.

* Sistema operativo Mac OS X, versión 10.7.
* Entorno de desarrollo Eclipse, versión 3.6.2.
* Librería Apache POI 3.7.
* Microsoft Word 2011 para Macintosh, versión 14.0.0.
* Microsoft PowerPoint 2011 para Macintosh, versión 14.0.0.
* MagicDraw UML, versión 17.0.1 beta 2.

### Recursos software para la implantación

Los recursos software empleados para la implantación son los siguientes.

* Sistema operativo Mac OS X.
* Librería Apache POI.
* J2SDK 6.0 o superior.

Parte II

ANÁLISIS DEL SISTEMA

# ESPECIFICACIÓN DE REQUISITOS

El Proyecto que se desea abordar consiste en el desarrollo de una librería Java escalable que permita la representación y gestión de *datasets*, así como la aplicación de operaciones complejas sobre los mismos. Cabe destacar, que se dotará de una interfaz gráfica de usuario para la prueba de los algoritmos implementados en la librería, no siendo este el objetivo principal del Proyecto, se analizará por tanto independientemente de la librería.

Es importante mencionar, que de aquí en adelante se hará referencia al Desarrollador y al Usuario, el primero se refiere a quien hace uso de la librería, el segundo a quien hace uso de la interfaz de usuario.

## Requisitos de información

Los requisitos en cuanto a la información a manejar son los siguientes:

* Requisito 1: información sobre el conjunto de datos a utilizar.
* Descripción: el sistema debe almacenar la información relativa al conjunto de datos sobre el que se desea actuar.
* Datos específicos: el *dataset* contendrá los siguientes datos:
* Nombre de los atributos
* Tipo de los atributos, pudiendo ser de tipo entero, real, categórico o de rango.
* Límite en cuanto al rango de cada uno de los atributos numéricos o categorías de los atributos categóricos.
* Secuencia de valores para cada atributo.
* Requisito 2: información sobre las estrategias
* Descripción: el sistema almacenará la información relativa a cada una de las estrategias que se implementen.
* Datos específicos
* Nombre de la estrategia.
* Clase Java a la que pertenece la estrategia.
* Parámetros (en caso de que existan).
* Descripción de la estrategia.

## Requisitos funcionales

A continuación se detallan los requisitos funcionales del software a desarrollar.

* Requisito 1: el desarrollador podrá cargar/guardar *datasets* cuatro tipos de *datasets,* que son, KEEL[5], Weka[6], CSV[7] y Excel[8].
* Requisito 2: el desarrollador podrá crear seis tipos de columnas, éstas son, entera, categórica, binaria, numérica, de rango y nominal, como entidad independiente al *dataset*.
* Requisito 3: el desarrollador podrá decidir cuáles serán los símbolos que representarán los valores vacíos, faltantes o nulos, así como su representación interna.
* Requisito 4: el desarrollador será capaz de modificar el contenido del *dataset*, esto es, metainformación, y el contenido que almacene, en este caso cada una de las columnas.
* Requisito 5: el desarrollador deberá ser capaz de modificar el contenido de una columna.
* Requisito 6: el desarrollador podrá consultar una columna del *dataset.*
* Requisito 7: el desarrollador será capaz de copiar a otro *dataset* su estructura, es decir, todo, excepto los valores de cada una de las columnas, también será capaz de copiar todo de forma integra, es decir, los valores de las columnas.
* Requisito 8: el desarrollador será capaz de realizar un intercambio entre dos columnas existentes en el *dataset*.
* Requisito 9: el desarrollador será capaz de aplicar cualquier estrategia de las definidas en la librería a cualquier *dataset*, esto es, discretización MDLP[4], eliminación de instancias repetidas, eliminación de un porcentaje de *dataset* y partición por el método KFolds[9].
* Requisito 10: el usuario podrá aplicar cualquier operación sobre columnas, sobre un rango determinado de columnas, esto es, discretizaciones, cambios de formato o reemplazamiento de valores.

Los diagramas de casos de uso representan como un usuario (actor) que opera con el sistema en desarrollo, además la forma, tipo y orden en la que los elementos de dicho sistema interactúan entre sí (operaciones o casos de uso). Los casos de uso se emplean para capturar el comportamiento deseado del sistema, sin tener que especificar cómo se implementa dicho comportamiento. Es decir, estos diagramas describen lo que hace el sistema desde el punto de vista de un observador externo, enfatizando el qué más que el cómo. Plantean escenarios en los que se muestra qué sucede cuando alguien interactúa con el sistema, proporcionando un resumen para una tarea u objetivo.

Dado que UML [29] no propone ningún modelo de plantilla para definir los casos de uso, en la Tabla 7.1 se definirá la plantilla que será utilizada en este proceso.

Tabla 7.1: Plantilla para definir casos de uso

Los casos de uso son requerimientos funcionales que describen, de una manera detallada, el comportamiento del sistema con los distintos actores que interactúan con él. Éstos no tienen por qué definir todos los requerimientos, pero representan el hilo conductor del sistema o aplicación, además de representar el carácter funcional de los objetivos.

|  |  |
| --- | --- |
| **Caso de uso:** Nombre del caso de uso | |
|  | |
| **Actores** | Actores que intervienen en el caso de uso. |
| **Descripción** | Descripción informal de los objetivos del caso de uso. |
| **Precondiciones** | Conjunto de condiciones que deben cumplirse para que el caso de uso se realice de forma correcta. |
| **Pasos a seguir** | Secuencia de pasos necesarios a seguir para la correcta realización del caso de uso. |

Mediante los casos de uso se guiará el proceso a pesar de que el objetivo no será realizar una descripción absoluta del sistema, sino más bien una descripción de los aspectos funcionales más importantes del mismo. Esta descripción al comienzo del desarrollo del software, permitirá empezar a comprender el sistema y llegar a los puntos más relevantes del mismo.

Según los objetivos y los requisitos que se han expuesto en los capítulos anteriores, en el sistema a desarrollar, existirán dos tipos de actores.

* Desarrollador: hará referencia a la persona que utilice la librería para desarrollar una aplicación más compleja, utilizando para ello las funciones que se le proporcionen.
* Usuario: será la persona que, a través de la interfaz de usuario, haga uso de las operaciones implementadas en la librería.

En los siguientes apartados se van a mostrar los casos de uso que se presentarán en el sistema software que se pretende desarrollar.

Para ello se realizará una división entre la librería y la interfaz de usuario.

### Subsistema Librería Predata

En la siguiente figura se muestra la funcionalidad global de la librería que se está analizando, mostrando su relación con el desarrollador externo.

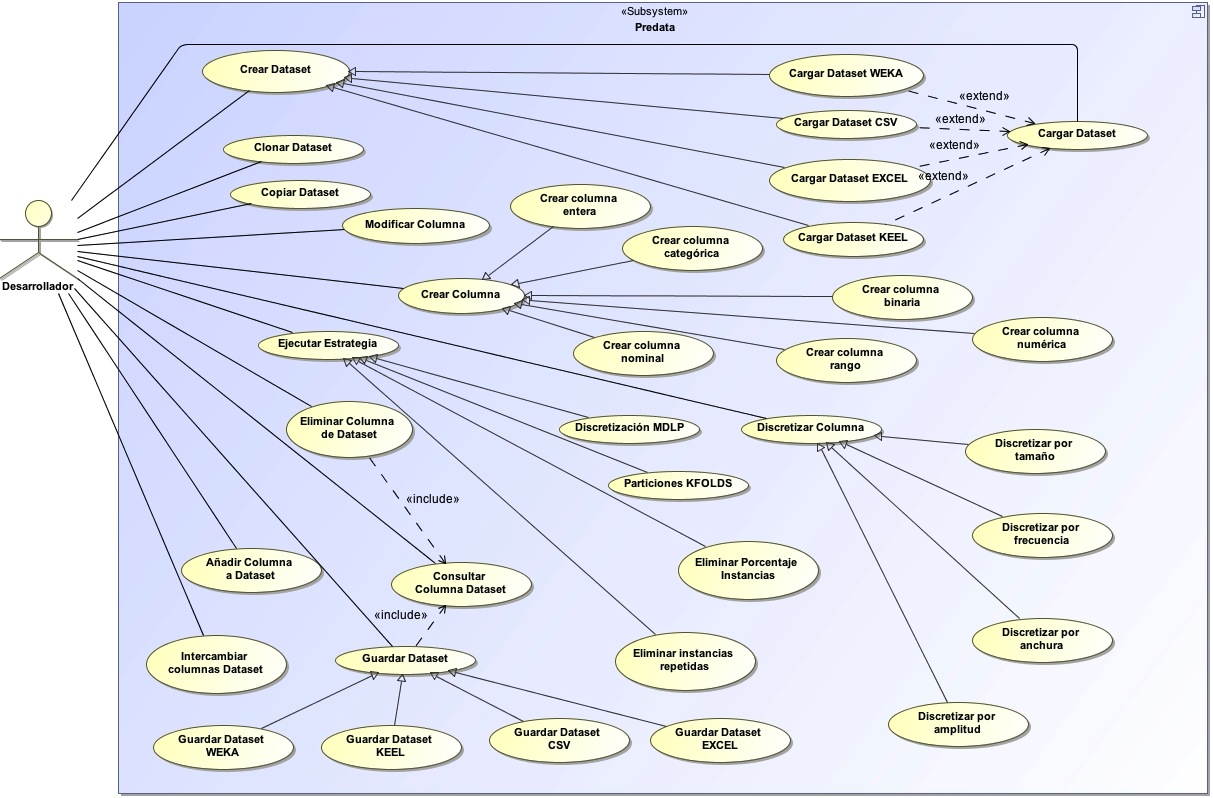


Figura 7.1: Subsistema Librería Predata

### Caso de uso 1: Crear *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Crear *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | Se crea un *dataset* vacío sin información, listo para ser usado. |
| **Precondiciones** | Ninguna. |
| **Pasos a seguir** | 1. Se definirá un objeto localizable dentro del código de la aplicación, para poder ser identificado. 2. El sistema llamará a la función constructora que inicializará las variables oportunas. 3. El sistema devolverá un objeto de tipo *dataset* con las características indicadas. |

Tabla 7.2: Caso de uso 1: Crear *Dataset*

Mediante este caso de uso, tal y como muestra el diagrama, el desarrollador será capaz de crear los cuatro tipos de *datasets* que se definirán en la librería, esto es, KEEL, Weka, CSV y Excel. Ya que el procedimiento de creación es común a los cuatro tipos, se detallará un solo caso de uso general para todos los formatos.

### Caso de uso 2: Cargar *Dataset*

Este caso de uso será el encargado de realizar la carga de un *dataset* de cualquier formato, desde el sistema de archivos, atendiendo al tipo de *dataset* y sus características respecto al formato del mismo, se especifica a continuación.

Tabla 7.3: Caso de uso 2: Cargar *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Cargar *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | La librería cargará un *dataset* desde el sistema de archivos. |
| **Precondiciones** | Ninguna. |
| **Pasos a seguir** | 1. Especificación de la ruta en el sistema de archivos del *dataset*. 2. Inicialización de las variables necesarias para la lectura. 3. El sistema realizará el análisis del contenido del *dataset.* 4. Adición del contenido a cada una de las columnas correspondientes. |

### Caso de uso 3: Crear Columna

Como se pone de manifiesto en la Figura 7.1 existe una generalización a partir de este caso de uso, la cual consta de seis casos de uso: Crear columna entera, Crear columna categórica, Crear columna binaria, Crear columna numérica, Crear columna rango y Crear columna nominal. En el que cada uno de ellos, llevan a cabo la creación de una columna del tipo que indica su propio nombre, no obstante se especificará un caso de uso común a todos ellos, ya que el proceso es similar en todos los casos.

Tabla 7.4: Caso de uso 3: Crear Columna

|  |  |
| --- | --- |
| **Caso de uso:** Crear Columna | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | La librería creará una nueva columna. |
| **Precondiciones** | Ninguna. |
| **Pasos a seguir** | 1. Se definirá un objeto localizable dentro del código de la aplicación, para poder ser identificado. 2. El sistema llamará a la función constructora según el tipo de columna que se crea. 3. El sistema devuelve una columna del tipo y características deseadas. |

### Caso de uso 4: Copiar *Dataset*

Tabla 7.5: Caso de uso 4: Copiar *Dataset*

Este caso de uso define las acciones llevadas a cabo por el sistema, para realizar la copia de un *dataset*, esto es, tanto estructura como contenido de cada una de las columnas que albergase. Cuando nos referimos a estructura, quiere decir, a los metadatos del *dataset,* es decir, cada una de las variables que contenga, así como cada una de las columnas del *dataset* no refiriéndonos al contenido de cada una de ellas.

|  |  |
| --- | --- |
| **Caso de uso:** Copiar *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | La librería copiará un *dataset* en otro. |
| **Precondiciones** | Se debe haber creado un *dataset* con anterioridad. |
| **Pasos a seguir** | 1. El sistema creará un nuevo *dataset* del mismo tipo que el origen. 2. Copia de la estructura del *dataset* origen al destino. 3. Copia del contenido de cada una de las columnas del *dataset* origen al destino. 4. El sistema devuelve la copia que es totalmente independiente al origen. |

### Caso de uso 5: Clonar *Dataset*

El caso de uso siguiente es similar al anterior, con la salvedad de que solamente se realiza una copia de la estructura del *dataset,* no del contenido del mismo.

Tabla 7.6: Caso de uso 5: Clonar *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Clonar *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | La librería copiará únicamente la estructura del *dataset* origen al destino.. |
| **Precondiciones** | Se debe haber creado un *dataset* con anterioridad. |
| **Pasos a seguir** | 1. El sistema creará un nuevo *dataset* del mismo tipo que el origen. 2. Copia de la estructura del *dataset* origen al destino. 3. El sistema devuelve la clonación que es totalmente independiente al origen. |

### Caso de uso 6: Modificar Columna

El caso de uso especificado en la Tabla 7.7 permite al desarrollador realizar una modificación del contenido de una columna.

Tabla 7.7: Caso de uso 6: Modificar Columna

|  |  |
| --- | --- |
| **Caso de uso:** Modificar Columna | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | Permite la modificación del contenido de una columna. |
| **Precondiciones** | Se debe haber creado una columna con anterioridad. |
| **Pasos a seguir** | 1. El sistema accede al contenido específico que se desea modificar. 2. Se modifica el contenido. 3. El sistema actualiza la columna, reflejando los cambios efectuados. |

### Caso de uso 7: Discretizar Columna

Este caso de uso, como se puede observar en la Figura 7.1 existe una generalización de este caso de uso, esto es, existen casos de usos derivados de este, Discretizar columna por tamaño, Discretizar columna por frecuencia, Discretizar columna por anchura y Discretizar columna por amplitud; en los que se lleva a cabo una discretización de la columna por el método que el nombre del caso de uso indica, ya que el procedimiento que realiza el sistema en cada uno de los casos es común, se realizará una descripción común a todos los demás casos, debido a la similitud del proceso.

|  |  |
| --- | --- |
| **Caso de uso:** Discretizar Columna | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | Se lleva a cabo una discretización de la columna. |
| **Precondiciones** | Se debe haber creado una columna de tipo numérico o entero con anterioridad. |
| **Pasos a seguir** | 1. El sistema llama al método correspondiente según el tipo de discretización que se realice. 2. El sistema lleva a cabo la discretización de la columna. 3. Se devuelve una nueva columna de tipo rango que contiene la discretización de la columna. |

Tabla 7.8: Caso de uso 7: Discretizar Columna

### Caso de uso 8: Añadir Columna a *Dataset*

Tabla 7.9: Caso de uso 8: Añadir Columna a *Dataset*

Mediante este caso de uso el desarrollador será capaz de añadir una columna previamente creada a un *dataset* que ya exista con anterioridad. Se define a continuación:

|  |  |
| --- | --- |
| **Caso de uso:** Añadir Columna a *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | Se añade una columna al *dataset* |
| **Precondiciones** | Deben existir un *dataset* y una columna previamente creados. |
| **Pasos a seguir** | 1. Se especifica el *dataset* donde se quiere introducir la columna y la posición que ocupará entre las ya existentes. 2. El sistema añade la columna a las ya existentes en el *dataset*. 3. El sistema refleja los cambios producidos en el *dataset* |

### Caso de uso 9: Eliminar Columna de *Dataset*

Este caso de uso, como su propio nombre indica, lleva a cabo la eliminación de una columna concreta en el *dataset*. Se especifica a continuación.

Tabla 7.10: Caso de uso 9: Eliminar Columna de *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Eliminar Columna de *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | Se elimina una columna del *dataset* |
| **Precondiciones** | Debe existir un *dataset* previamente creado y este debe contener columnas. |
| **Pasos a seguir** | 1. Se especifica el *dataset* del que se quiere eliminar la columna. 2. Se especifica la posición de la columna a eliminar. 3. El sistema elimina la columna indicada de las existentes. 4. El sistema refleja los cambios producidos en el *dataset*. |

### Caso de uso 10: Consultar Columna *Dataset*

El desarrollador a través de este caso de uso será capaz de obtener una columna del *dataset,* para realizar modificaciones en ella o simplemente realizar consultas sobre su contenido. El caso de uso se especifica a continuación.

Tabla 7.11: Caso de uso 10: Consultar Columna *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Consultar Columna *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | El sistema devuelve la referencia a una columna del *dataset*. |
| **Precondiciones** | Debe existir un *dataset* previamente creado y este contener columnas. |
| **Pasos a seguir** | 1. Especificación del índice donde se encuentra la columna deseada. 2. El sistema devuelve la referencia de la columna solicitada, en caso de que existiese el índice especificado. |

### Caso de uso 11: Intercambiar columnas *Dataset*

Tabla 7.12: Caso de uso 11: Intercambiar columnas *Dataset*

Este caso de uso permite al desarrollador, realizar el intercambio de la posición de dos columnas en el *dataset*. Los pasos a seguir se especifican en la siguiente tabla.

|  |  |
| --- | --- |
| **Caso de uso:** Intercambiar columnas *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | Se intercambia la posición de dos columnas del *dataset.* |
| **Precondiciones** | Debe existir un *dataset* y este debe contener al menos dos columnas. |
| **Pasos a seguir** | 1. Se especifica el *dataset* sobre el que se desea actuar. 2. Se especifican las dos columnas que se desean intercambiar. 3. El sistema refleja los cambios efectuados en el *dataset*. |

### Caso de uso 12: Guardar *Dataset*

La Tabla 7.13 muestra el Caso de uso 12, tal y como se puede observar en la Figura 7.1 de este mismo capítulo, existe una generalización del mismo, esto es, Guardar *Dataset* Weka, Guardar *Dataset* KEEL, Guardar *Dataset* CSV y Guardar *Dataset* Excel. En los que se realiza el guardado del *dataset* en el formato que indica el propio nombre del caso de uso. Como ocurre en los anteriores casos de uso en los que existe una generalización, el procedimiento es común a cada formato, se especificará un caso de uso general a los cuatro formatos.

Tabla 7.13: Caso de uso 12: Guardar *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Guardar *Dataset* | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | Se realiza el guardado al sistema de archivos del *dataset*. |
| **Precondiciones** | Deben existir un *dataset* previamente creado y que contenga columnas. |
| **Pasos a seguir** | 1. Especificación de la ruta donde se guardará el *dataset*. 2. El sistema escribirá la cabecera del *dataset* si es necesario. 3. Escritura del contenido de cada una de las columnas. 4. Se cierra el archivo asociado al *dataset*. |

### Caso de uso 13: Ejecutar Estrategia

Este caso de uso es el que permitirá al desarrollador aplicar sobre un *dataset* un algoritmo de los definidos en la librería. Se observa que existe una generalización del mismo, esto es, Discretización MDLP, Eliminar instancias repetidas, Eliminar porcentaje instancias y Particiones KFolds. En los que se aplica cada uno de los algoritmos que indica el propio nombre del caso de uso. El procedimiento de aplicación de cada uno de ellos es común, por lo que se realiza la especificación general de los mismos. La secuencia de pasos que realiza el sistema se especifica en la Tabla 7.14.

Tabla 7.14: Caso de uso 13: Ejecutar Estrategia

|  |  |
| --- | --- |
| **Caso de uso:** Ejecutar Estrategia | |
|  | |
| **Actores** | Desarrollador. |
| **Descripción** | Se lleva a cabo la ejecución de un algoritmo sobre un *dataset*. |
| **Precondiciones** | Permite ejecutar un algoritmo que se aplica sobre *datasets* de los definidos en la librería. |
| **Pasos a seguir** | 1. Se especifica el *dataset* sobre el que se aplicará el algoritmo. 2. El sistema inicializa las variables necesarias para la ejecución del algoritmo. 3. Se lleva a cabo la ejecución del algoritmo. 4. El sistema devuelve el resultado de aplicar el algoritmo sobre el *dataset*. |

### Subsistema Interfaz de usuario

En la siguiente figura se muestra la funcionalidad que se ofrece en la interfaz de usuario, mostrando su relación con el usuario.

### Macintosh HD:Users:psylock:Desktop:interfaz.jpg

Figura 7.2: Subsistema Interfaz de usuario

### Caso de uso 14: Crear Proyecto

Mediante este caso de uso el desarrollador podrá crear un nuevo proyecto, en el que podrá hacer uso de todas las operaciones de la interfaz, y será desde donde se realizará el acceso a los *datasets* en uso por la interfaz.

|  |  |
| --- | --- |
| **Caso de uso:** Crear Proyecto | |
|  | |
| **Actores** | Usuario |
| **Descripción** | El usuario creará un nuevo proyecto para poder realizar operaciones sobre datasets. |
| **Precondiciones** | No existen. |
| **Pasos a seguir** | 1. El usuario pulsa con el botón derecho en la ventana del explorador de proyectos, seleccionando Nuevo Proyecto. 2. El usuario elije el nombre del proyecto y donde se ubicará en el sistema de archivos. 3. El proyecto es añadido a la lista de proyectos en el explorador de proyectos. |

Tabla 7.15: Caso de uso 14: Crear Proyecto

### Caso de uso 15: Crear *Dataset* Nuevo

El usuario será capaz mediante este caso de uso de crear un *dataset* vacío en un proyecto ya existente. Su especificación es la siguiente.

Tabla 7.16: Caso de uso 15: Crear *Dataset* Nuevo

|  |  |
| --- | --- |
| **Caso de uso:** Crear *Dataset* Nuevo | |
|  | |
| **Actores** | Usuario |
| **Descripción** | Permite la creación de un *dataset* vacío. |
| **Precondiciones** | Es necesario que exista un proyecto creado previamente. |
| **Pasos a seguir** | 1. El usuario pulsa con el botón derecho sobre el proyecto donde se desea ubicar el dataset. 2. Se selecciona la opción Nuevo Archivo. 3. El usuario elige el nombre del *dataset* especificando su extensión. |

### Caso de uso 16: Importar *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Importar *Dataset* | |
|  | |
| **Actores** | Usuario |
| **Descripción** | Permite la importación de un *dataset* existente en el sistema de archivos a un proyecto. |
| **Precondiciones** | Es necesario que exista un proyecto creado previamente. |
| **Pasos a seguir** | 1. El usuario pulsa con el botón derecho sobre el proyecto donde se desea ubicar el dataset. 2. Se selecciona la opción Importar 3. El usuario busca el *dataset* en el sistema de archivos. 4. El *dataset* es añadido al proyecto seleccionado. |

El usuario podrá mediante este caso de uso, importar un *dataset* ya existente en el sistema de archivos, a un proyecto determinado. Se detalla a continuación:

Tabla 7.17: Caso de uso 16: Importar *Dataset*

### Caso de uso 17: Exportar *Dataset*

Tabla 7.18: Caso de uso 17: Exportar *Dataset*

Se le permite al usuario mediante el caso de uso especificado en la Tabla 7.18 exportar un *dataset* desde un proyecto a cualquier ubicación en el sistema de archivos. Se detalla a continuación.

|  |  |
| --- | --- |
| **Caso de uso:** Exportar *Dataset* | |
|  | |
| **Actores** | Usuario |
| **Descripción** | Permite exportar un *dataset* existente en un proyecto al sistema de archivos. |
| **Precondiciones** | Es necesario haber creado/importado un *dataset* en un proyecto. |
| **Pasos a seguir** | 1. El usuario pulsa con el botón derecho sobre el *dataset* en cuestión desde el explorador de proyectos. 2. El usuario hace clic sobre la opción exportar. 3. El usuario elige la ubicación del *dataset*  y lo guarda. |

### Caso de uso 18: Cargar *Dataset* en Memoria

Mediante este caso de uso se permite al usuario cargar un *dataset* determinado en memoria, para poder aplicarle operaciones o estrategias. Se detalla a continuación.

|  |  |
| --- | --- |
| **Caso de uso:** Cargar *Dataset* en Memoria | |
|  | |
| **Actores** | Usuario. |
| **Descripción** | Permite cargar un *dataset* en memoria para poder manipularlo. |
| **Precondiciones** | Es necesario que exista un *dataset* en un proyecto |
| **Pasos a seguir** | 1. El usuario hace doble clic sobre el *dataset*. 2. En el menú Archivo, el usuario hace clic en Cargar. 3. El usuario selecciona el formato del *dataset* y hace clic en Cargar. |

Tabla 7.19: Caso de uso 18: Cargar *Dataset* en Memoria

### Caso de uso 19: Aplicar operación sobre columna(s)

Este caso de uso representa la acción por la cual se le permite al usuario aplicar una operación sobre una o varias columnas del *dataset* previamente cargado. Se detalla en la siguiente tabla.

|  |  |
| --- | --- |
| **Caso de uso:**  Aplicar operación sobre columna(s) | |
|  | |
| **Actores** | Usuario. |
| **Descripción** | Permite aplicar una operación sobre una o varias columnas del *dataset*. |
| **Precondiciones** | Es necesario haber cargado previamente el *dataset* para su manipulación. |
| **Pasos a seguir** | 1. El usuario elige una operación en la parte inferior de la ventana. 2. El usuario introduce el rango de columnas sobre el que se aplicará la operación. 3. El usuario introducirá los demás parámetros propios de cada operación. 4. El usuario hace clic en el botón Ejecutar. |

Tabla 7.20: Caso de uso 19: Aplicar operación sobre columna(s)

### Caso de uso 20: Escribir *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Escribir *Dataset* | |
|  | |
| **Actores** | Usuario. |
| **Descripción** | Permite guardar un *dataset* previamente cargado. |
| **Precondiciones** | Es necesario haber cargado previamente el *dataset*. |
| **Pasos a seguir** | 1. El usuario hace clic en el menú Archivo en Guardar. 2. El usuario selecciona el formato de destino y el nombre del mismo. 3. El usuario hace clic sobre Guardar. |

Mediante la utilización de este caso de uso, se permite guardar el *dataset* previamente cargado, y manipulado si es el caso, en cualquier formato soportado por la librería. Se detalla a continuación:

Tabla 7.21: Caso de uso 20: Escribir *Dataset*

### Caso de uso 21: Aplicar estrategia sobre *Dataset*

Mediante este caso de uso se especifica la aplicación de una estrategia (algoritmo) sobre un *dataset* completo previamente cargado. Se detalla a continuación.

Tabla 7.22: Caso de uso 21: Aplicar estrategia sobre *Dataset*

|  |  |
| --- | --- |
| **Caso de uso:** Aplicar estrategia sobre *Dataset* | |
|  | |
| **Actores** | Usuario. |
| **Descripción** | Permite aplicar un algoritmo sobre un dataset completo |
| **Precondiciones** | Es necesario haber cargado previamente el *dataset*. |
| **Pasos a seguir** | 1. El usuario selecciona una estrategia en la pestaña de estrategias de la interfaz. 2. El usuario introduce los parámetros para el algoritmos, si los hubiera. 3. El usuario hace clic en Ejecutar. |

## Requisitos no funcionales

Los requisitos no funcionales son aquellos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino que permiten especificar características de arquitectura lógica, infraestructura, interfaz, seguridad, rendimiento, disponibilidad, integración y portabilidad.

Los requisitos no funcionales son los que se detallan a continuación.

* Requisito 1: La librería debe ser intuitiva y eficaz, de forma que se pueda acceder a toda la funcionalidad rápidamente y de forma robusta, informando al usuario de lo que ocurre en cada momento.
* Requisito 2: El sistema deberá responder contra posibles errores derivados del funcionamiento del propio sistema.
* Requisito 3: El sistema deberá ser eficiente, es decir, llevar a cabo su ejecución de forma que el tiempo empleado sea el mínimo y con la mínima utilización de recursos.
* Requisito 4: Todos los mensajes de error del sistema deben ser significativos, de manera que se incluya un texto descriptivo.
* Requisito 5: El sistema deberá ser escalable, es decir, su funcionalidad podrá ser extendida sin que afecte a ningún módulo ya desarrollado, o en el peor de los casos afectará al mínimo número de módulos posible.

# DESCRIPCIÓN DE LA INFORMACIÓN

En este capítulo se va a mostrar la información que maneja el sistema. Para ello nos adentraremos en la estructura interna que deberá tener para poder cumplir con los requisitos del sistema. Posteriormente, se analizarán los módulos y clases necesarias para manejar dicha información, la cual ha sido extraída del estudio del problema.

## Formato de los datos de entrada y salida

Para la aplicación de cualquier operación implementada en la librería, es necesario la manipulación de diferentes tipos de ficheros de entrada, en este caso, los *datasets.* Se implementarán cuatro formatos distintos: KEEL[5], Weka[6], CSV[7] y Excel[8].

### *Dataset* KEEL

Los ficheros KEEL que se utilizarán como *dataset* de entrada, tiene el siguiente formato.

* Nombre del conjunto de datos (obligatorio):

@relation “nombre”

* Atributos (obligatorio):

@attribute “nombre” integer [min, max]

@attribute “nombre” real [min, max]

@attribute “nombre” interval [min, max]

@attribute “nombre” {“valor1”, “valor2”, valor3”, …, “valorN”}

@attribute “nombre” string

La primera definición correspondería a una columna de tipo numérico entero, la segunda a una columna de tipo real, la siguiente a una columna de tipo rango, la penúltima a una columna de tipo categórico, en la que se indican la lista de valores posibles para esa columna; y por último una columna de tipo nominal.

* Datos (obligatorio): Se sigue el orden en el que se definieron los atributos:

@data

X11, ··········, X1n

· ·

X22, ·········, X2n

Cada instancia vendrá representada por una línea después de la palabra clave **@data,** dentro de cada instancia, cada valor perteneciente a cada columna viene separado por comas.

Si se necesita indicar un valor vacío, nulo o faltante, podrá ser cualquier valor, la librería tiene la capacidad para poder leer teniendo en cuenta el valor especificado como inválido.

Estos ficheros se almacenarán, por defecto, con la extensión *dat*. Un ejemplo de este tipo de ficheros podría ser el siguiente.

**@relation** diagnose

**@attribute** name string

**@attribute** id integer [0,1000]

**@attribute** temperature interval [34,42]

**@attribute** continent {european, american, asian, african, other}

**@data**

Jack, 0, 36, american

Jin, 1, 36.5, asian

Kate, 2, 36, other

Ecko, 3, 37, african

Charlie, 4, 38, european

### *Dataset* CSV

El formato de ficheros CSV, es usado normalmente para el intercambio de datos entre diferentes aplicaciones.

Estos ficheros presentan el siguiente formato.

* Nombres de las columnas de los datos (puede ir al final o al principio, no obligatorio)
* Los datos de cada columna (al final o al principio, es obligatorio), que pueden ser de tres tipos:
* Enteros
* Reales
* Cadenas
* Los datos se encuentran separados por comas.

X11, ··········, X1n

· ·

X22, ·········, X2n

Al igual que en el *dataset* KEEL, cada instancia viene representada por una línea y cada valor es separado mediante una coma.

Estos ficheros se almacenarán, por defecto, con la extensión *csv*. Un ejemplo de este tipo de ficheros puede ser el siguiente.

name, id, temperature, continent

Jack, 0, 36, american

Jin, 1, 36.5, asian

Kate, 2, 36, other

Ecko, 3, 37, african

Charlie, 4, 38, european

### *Dataset* Weka

Los archivos ARFF que se usarán como entrada, presentan idéntico formato al definido por KEEL, con la salvedad de que la cabecera se define de la siguiente forma.

@attribute “nombre” numeric

@attribute “nombre” {“valor1”, “valor2”, valor3”, …, “valorN”}

@attribute “nombre” string

La diferencia que presentan este tipo de archivos con KEEL, es que para definir tipos numéricos, se usa la palabra clave *numeric* y no se define el intervalo del mismo. Tampoco existe el tipo de atributo *interval*, que define rangos numéricos. Un ejemplo de este tipo de *dataset* puede ser.

@relation diagnose

@attribute name string

@attribute id numeric

@attribute temperature numeric

@attribute continent {european, american, asian, african, other}

@data

Jack, 0, 36, american

Jin, 1, 36.5, asian

Kate, 2, 36, other

Ecko, 3, 37, african

Charlie, 4, 38, european

### *Dataset* Excel

Este tipo de *dataset* es un tipo de archivo propietario de Microsoft, el cual, su estructura no puede definir, ya que se trata de un tipo de archivo binario y no texto como los anteriores.

Estos ficheros se almacenarán con las extensiones *xls* o *xlsx* al ser archivos binarios deben ser visualizados con programas como Microsoft Excel, a continuación se muestra un ejemplo de un *dataset* cargado en Microsoft Excel.

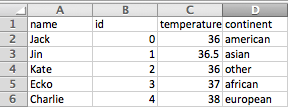


Figura 8.1: Ejemplo de un *dataset* cargado en Microsoft Excel

## Fichero definición de estrategias

El lenguaje de marcado XML busca dar solución al problema de expresar información estructurada de un modo más abstracto y reutilizable, en contraposición a la anarquía existente históricamente a la hora de almacenar la información de los programas. El lenguaje XML se compone de partes bien definidas que a su vez se componen de otras partes. Éstas se llaman elementos y se añaden mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Las etiquetas tienen la forma *<tag></tag>*, donde *tag* es el nombre que recibe el elemento representado. Todas las etiquetas deben contener su correspondiente etiqueta de cierre o en su defecto representarse de la siguiente forma *<tag />*.

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento.

Existe una etiqueta principal para definir cada estrategia, esta es, *<Algorithm>*, donde se definirán las características del algoritmo en cuestión, contendrá las siguientes etiquetas:

* Nombre: indica el nombre de la estrategia, se especifica con *<name></name>.*
* Clase: indica la clase asociada a la estrategia, se especifica mediante *<class></class>.*
* Parámetros: lista de parámetros para la ejecución de la estrategia, es posible especificarlos mediante la etiqueta *<parameters></parameters>.*
* Descripción: breve descripción de la finalidad del algoritmo, para su especificación es necesario la utilización de *<description></description>.*

Nótese que es necesario que dentro de cada etiqueta, su valor venga encerrado entre *<value> …. </value>.*

No obstante se muestra a continuación un ejemplo de cómo definir una estrategia usando el formato XML definido anteriormente.

<algorithm>

<!-- Defines the name of the strategy -->

<name>

<value>

Remove Percentage

</value>

</name>

<!-- Defines the class associated with the strategy -->

<class>

<value>

RemovePercentage

</value>

</class>

<!-- Defines the parameters of the strategy-->

<parameters>

<value>

Mode of removal,Percentage to remove

</value>

</parameters>

<!-- Defines a description for the strategy -->

<description>

<value>

Remove percentage from a dataset

according to three modes:

Mode 0: From the start of dataset

Mode 1: From the end of dataset

Mode 2: Randomly on whole dataset

</value>

</description>

</algorithm>

# ANÁLISIS FUNCIONAL

## Descripción arquitectónica

Una arquitectura de software, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información.

Una arquitectura software se selecciona y diseña con base en objetivos y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como la mantenibilidad, auditabilidad, flexibilidad e interacción con otros sistemas de información. Las restrcciones son aquellas limitaciones derivadas de tecnologías disponibles para implementar sistemas de información.

La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar qué computadora tendrá asignada cada tarea. Aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

La librería está compuesta por las siguientes capas:

* *Predata,* es la capa que proporciona la funcionalidad completa de la librería, su funcionalidad se apoya en la descrita en la Sección 7.2 de este mismo documento.
* *Interfaz de usuario*, es la capa que ofrece la interfaz de usuario para la prueba de los algoritmos desarrollados en la capa *Predata.*

## Componentes del sistema

En este apartado se analizarán los componentes esenciales implicados en el sistema, más tarde, en la especificación del diseño de la aplicación se profundizará más en el contenido de cada paquete por separado, esta sección está presente en el Capítulo 10 de este mismo documento.

Los componentes fundamentales de cada paquete y las relaciones existentes entre ellos se especifican en la siguiente figura.

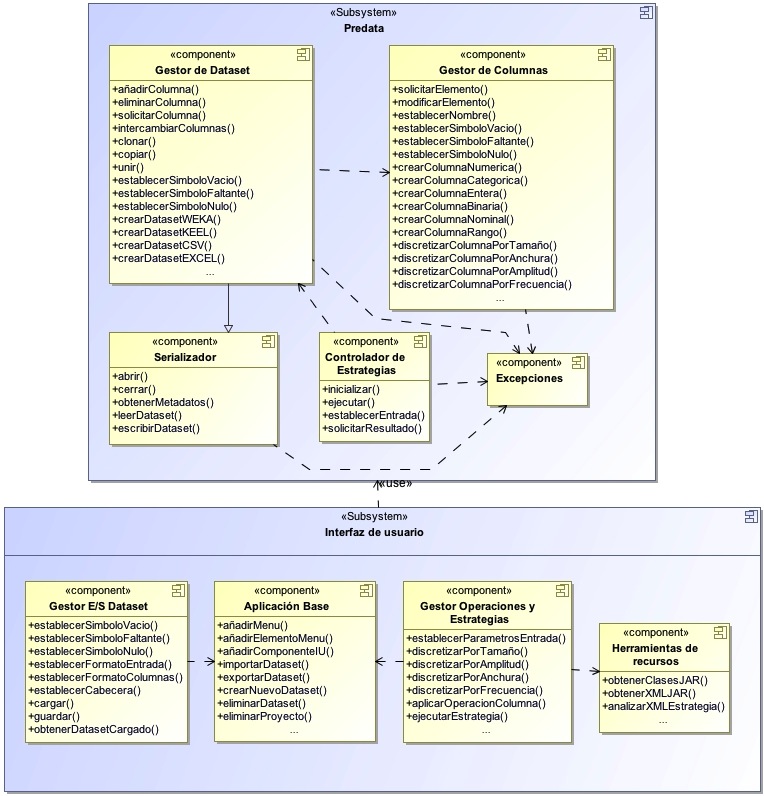


Figura 9.1: Arquitectura del sistema

### Componente Gestor de *Dataset*

El componente Gestor de *Dataset*, representa la entidad encargada de la gestión del contenido del *dataset* y de sus metadatos, mediante este componente el sistema podrá realizar las siguientes acciones:

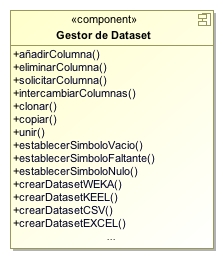


Figura 9.2: Componente Gestor de *Dataset*

* *añadirColumna():* añadirá una columna a la estructura del *dataset*.
* *eliminarColumna():* eliminará una columna determinada del *dataset.*
* *solicitarColumna():* devolverá la columna solicitada.
* *intercambiarColumnas():* realizará un intercambio entre dos columnas del *dataset.*
* *clonar():* realizará una copia únicamente de la estructura del *dataset,* no del contenido de cada una de las columnas.
* *copiar():* realizara una copia completa del *dataset* a otro *dataset.*
* *unir():* realiza la unión de dos *datasets* con el mismo número y tipo de columnas.
* *establecerSimboloVacio():* establece el símbolo por el cual se representan los valores vacíos.
* *establecerSimboloFaltante():* establece el símbolo por el cual se representan los valores faltantes.
* *establecerSimboloNulo():* establece el símbolo por el cual se representan los valores nulos.
* *crearDatasetWEKA():* este método crea un *dataset* en formato Weka descrito anteriormente.
* *crearDatasetKEEL():* mediante la utilización de este método el desarrollador podrá crear un *dataset* del tipo KEEL.
* *crearDatasetCSV():* a través de este método es posible crear un *dataset* en formato *csv* anteriormente detallado.
* *crearDatasetEXCEL():* la utilización de este método posibilita la creación del *dataset* Excel especificado el capítulo 8.1.4.

### Componente Gestor de Columnas

El componente Gestor de columnas es el encargado de aportar al sistema las funciones necesarias para el manejo de columnas, mediante este componente el sistema será capaz de llevar a cabo las siguientes operaciones:

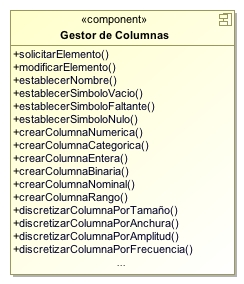


Figura 9.3: Componente Gestor de Columnas

* *solicitarElemento():* devuelve un elemento determinado de la columna.
* *modificarElemento():* modifica un elemento determinado en la columna.
* *establecerNombre():* establece el nombre del atributo (de la columna).
* *establecerSimboloVacio():* establece el símbolo que se almacenará para representar un valor vacío.
* *establecerSimboloFaltante():* establece el símbolo que se usará para representar un valor faltante en la columna.
* *establecerSimboloNulo():* determina el símbolo que servirá para identificar un valor nulo en la columna.
* *crearColumnaNumerica():* crea una columna de tipo numérico.
* *crearColumnaCategorica ():* lleva a cabo la creación de una columna del tipo categórico.
* *crearColumnaEntera():* crea una columna de tipo entero.
* *crearColumnaBinaria():* este método devuelve una nueva columna de tipo binario.
* *crearColumnaNominal():* devuelve una columna de tipo nominal.
* *crearColumnaRango():* crea una columna de tipo rango.
* *discretizarPorTamaño():* lleva a cabo una discretización por tamaño de partición
* *discretizarPorAnchura():* realiza una discretización por anchura de partición.
* *discretizarPorAmplitud():* ejecuta una discretización sobre una columna por amplitud de particiones
* *discretizarPorFrecuencia():* aplica una discretización por frecuencia de valores en las particiones.

### Componente Serializador

El componente Serializador será el encargado de serializar un *dataset*, esto es, escribirlo al sistema de archivos, o recuperarlo desde él, mediante el uso de este componente el sistema obtendrá las siguientes operaciones:

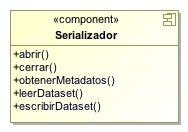


Figura 9.4: Componente Serializador

* *abrir():* realiza la apertura del archivo en el sistema de archivos.
* *cerrar():* lleva a cabo del cierre de un archivo previamente abierto.
* *obtenerMetadatos():* realiza la lectura de los metadatos que pudiera contener el *dataset*, normalmente en la cabecera del mismo.
* *leerDataset():* lleva a cabo la recuperación de todos los valores del *dataset* volcándolos en las columnas correspondientes.
* *escribirDataset():* escribe el *dataset* almacenado al sistema de archivos, escribiendo su cabecera (si fuese necesario) y los valores de cada una de las columnas.

### Componente Controlador de Estrategias

Este componente, es el que llevará a cabo la gestión de las estrategias, mediante este componente, se podrá hacer uso de cada uno de los algoritmos definidos para *dataset* en la librería, aporta la siguiente funcionalidad al sistema:

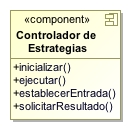


Figura 9.5: Componente Controlador de Estrategias

* *inicializar():* inicializará el algoritmo para poder ejecutarlo.
* *ejecutar():* llevará a cabo la ejecución del algoritmo.
* *establecerEntrada():* este método establece el *dataset* de entrada al que se le aplicará el algoritmo.
* *solicitarResultado():* este método devuelve el resultado de aplicar el algoritmo sobre la entrada.

### Componente Excepciones

Mediante este componente, el sistema será capaz de informar al desarrollador que use la librería de situaciones anómalas que pudieran producirse durante la ejecución de alguna función de la librería. Este albergará las clases necesarias para cada tipo de excepción.



Figura 9.6: Componente Excepciones

Se deberá realizar el control de las siguientes excepciones:

* Un valor no se ha podido añadir a una columna.
* Un valor está fuera de los rangos permitidos para una columna.
* No existe la categoría especificada en una columna de tipo categórico.
* No se ha realizado una especificación correcta del tipo de las columnas a la hora de cargar un *dataset.*

### Componente Gestor E/S *Dataset*

El componente *DatasetStrategy*, perteneciente a la interfaz de usuario, aporta a la interfaz los controles necesarios para manejar la entrada y salida de los *dataset*, además de su edición en la propia interfaz. La funcionalidad que aporta este componente al sistema es la siguiente:

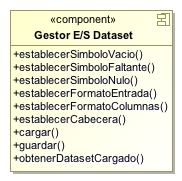


Figura 9.7: Componente Gestor E/S *Dataset*

* *establecerSimboloVacio():* establece el símbolo por el cual se representan los valores vacíos en el *dataset* a la lectura del mismo.
* *establecerSimboloFaltante():* establece el símbolo que se usará para representar los valores faltantes en el *dataset* para la lectura.
* *establecerSimboloNulo():* define el símbolo que sirve para identificar un valor nulo en el *dataset* para su lectura.
* *establecerFormatoEntrada():* establece el formato del *dataset* de entrada.
* *establecerFormatoColumnas():* define los tipos de columna que contiene el *dataset* si se trata de un formato sin cabecera.
* *establecerCabecera():* define la cabecera para un *dataset* proveniente de un formato que no posea cabecera.
* *cargar():* realiza la carga en memoria del *dataset*.
* *guardar():* realiza el volcado del contenido del *dataset* al sistema de archivos.

### Componente Aplicación Base

Se trata del componente que define la base para la interfaz de usuario, es donde se añadirán todos los elementos que conformarán la plataforma completa. Aporta la siguiente funcionalidad:

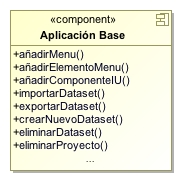


Figura 9.8: Componente Aplicación Base

* *añadirMenu():* añade un menú a la barra de herramientas de la aplicación.
* *añadirElementoMenu():* añade un elemento a un menú previamente creado, que puede ser a su vez otro menú.
* *añadirComponenteIU():* añade un componente funcional a la interfaz base, como podría ser un editor.
* *importarDataset():* importa un *dataset* desde el sistema de archivos al espacio de trabajo de la aplicación.
* *exportarDataset():* exporta un *dataset* desde el espacio de trabajo de la aplicación al sistema de archivos.
* *crearNuevoDataset():* crea un nuevo *dataset* vacío.
* *eliminarDataset():* elimina un *dataset* del espacio de trabajo.
* *eliminarProyecto():* elimina un proyecto del espacio de trabajo.

### Componente Gestor operaciones y estrategias

A través de este componente, la interfaz de usuario será capaz de realizar la gestión de las operaciones aplicables sobre columnas, además de los algoritmos disponibles en la librería para aplicar a *datasets*. Aporta al sistema la siguiente funcionalidad:

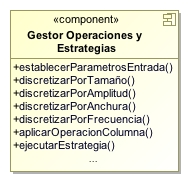


Figura 9.9: Componente Gestor operaciones y estrategias

* *establecerParametrosEntrada():* método que establece los parámetros de entrada para una operación.
* *discretizarPorTamaño():* lleva a cabo una discretización por tamaño de partición.
* *discretizarPorAnchura():* realiza una discretización por anchura de partición.
* *discretizarPorAmplitud():* ejecuta una discretización sobre una columna por amplitud de particiones.
* *discretizarPorFrecuencia():* aplica una discretización por frecuencia de valores en las particiones.
* *ejecutarEstrategia():* método que lleva a cabo la ejecución de una estrategia determinada.

### Componente Herramientas de recursos

Este componente representa un conjunto de funciones que se añaden a la funcionalidad base de la interfaz de usuario, aporta funciones útiles para obtener información acerca del entorno de ejecución, realiza las siguientes acciones:

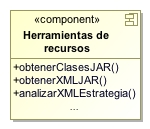


Figura 9.10: Componente Herramientas de recursos

* *obtenerClasesJAR():* obtiene las clases que contiene un archivo JAR que se encuentre en el entorno de ejecución.
* *obtenerXMLJAR():* obtiene los archivos XML que contiene un archivo JAR situado en el entorno de ejecución.
* *analizarXMLEstrategia():* método útil para la carga dinámica de estrategias en la interfaz, analiza información como, nombre, parámetros y descripción.

## Descripción del comportamiento

La descripción del comportamiento se llevará a cabo a través de los diagramas de secuencia. Los diagramas de secuencia describen la colaboración existente entre los objetos que componen el sistema.

Los diagramas de secuencia se obtienen de forma prácticamente automática a partir de los casos de uso. Esto quiere decir que cada caso de uso tendrá un diagrama de secuencia asociado. Por esta razón se describirán en el mismo orden que los casos de uso expuestos con anterioridad, para tener una mayor compresión. A continuación se exponen y describen los diagramas de secuencia resultantes de los casos de uso descritos previamente.

### Diagrama de secuencia 1: Crear *Dataset*

La operación consiste en invocar a la función constructora del *Dataset* deseado con la información necesaria para su creación.

La secuencia temporal de los pasos realizados es la siguiente:

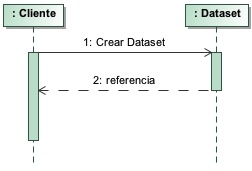


Figura 9.11: Diagrama de secuencia 1: Crear *Dataset*

### Diagrama de secuencia 2: Cargar *Dataset*

Esta operación consiste, en una vez ya creado un *dataset*, cargar el contenido desde el sistema de archivos. Su secuencia temporal se indica en la siguiente figura.

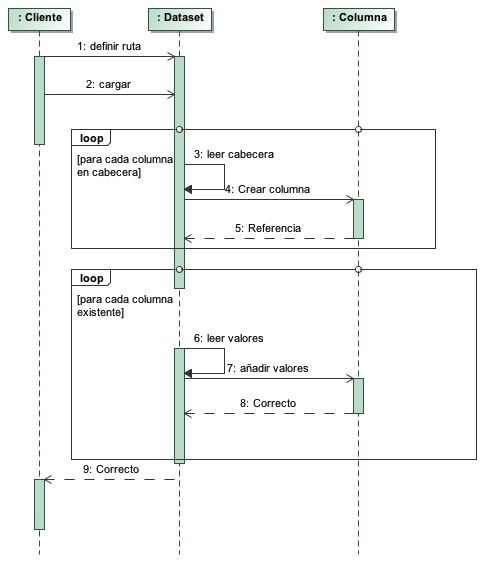


Figura 9.12: Diagrama de secuencia 2: Cargar *dataset*

### Diagrama de secuencia 3: Crear Columna

Esta secuencia de pasos permitirá al desarrollador la creación de una nueva columna, se detalla en la siguiente figura.

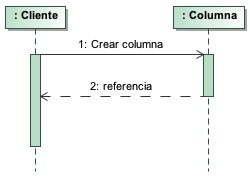


Figura 9.13: Diagrama de secuencia 3: Crear Columna

### Diagrama de secuencia 4: Copiar *Dataset*

Esta operación consiste a la descrita en el caso de uso 7.2.4, la secuencia de pasos se describe a continuación.

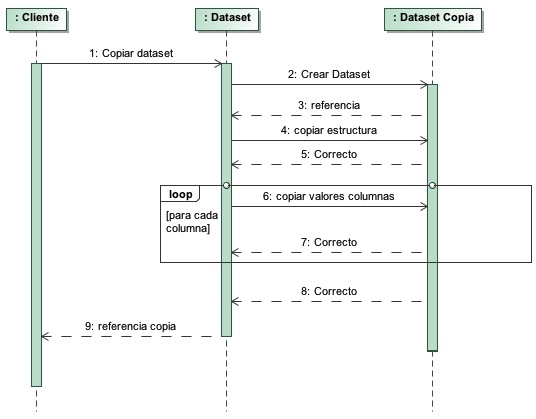


Figura 9.14: Diagrama de secuencia 4: Copiar *Dataset*

### Diagrama de secuencia 5: Clonar *Dataset*

Mediante este diagrama de secuencia, es posible realizar la clonación de la estructura de un *dataset.*

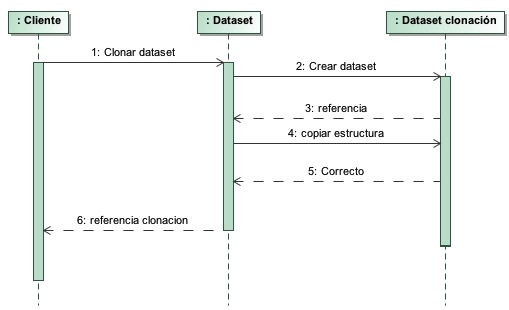


Figura 9.15: Diagrama de secuencia 5: Clonar *Dataset*

### Diagrama de secuencia 6: Modificar Columna

Esta operación permitirá al desarrollador, modificar una columna ya existente. La secuencia de pasos se especifica en la Figura 9.15.

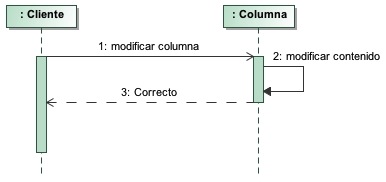


Figura 9.16: Diagrama de secuencia 6: Modificar Columna

### Diagrama de secuencia 7: Discretizar Columna

Mediante este diagrama de secuencia, se le permitirá al desarrollador llevar a cabo la discretización de una columna. Su descripción se realiza en la siguiente figura.

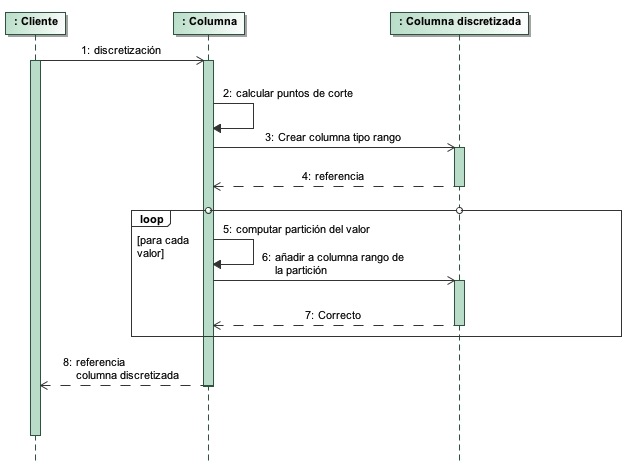


Figura 9.17: Diagrama de secuencia 7: Discretizar Columna

### Diagrama de secuencia 8: Añadir Columna a *Dataset*

En este diagrama se describe la secuencia de pasos a seguir para añadir una columna ya existente a un *dataset* previamente creado.

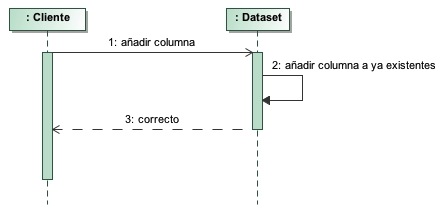


Figura 9.18: Diagrama de secuencia 8: Añadir Columna a *Dataset*

### Diagrama de secuencia 9: Eliminar Columna *Dataset*

Mediante este diagrama de secuencia el desarrollador será capaz de llevar a cabo correctamente, la eliminación de una columna existente en un *dataset.* Se describe en la siguiente figura.

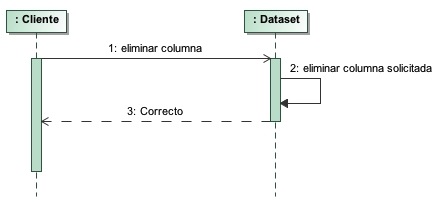


Figura 9.19: Diagrama de secuencia 9: Eliminar Columna D*ataset*

### Diagrama de secuencia 10: Consultar Columna *Dataset*

Esta secuencia de pasos describe la manera de realizar la consulta de una columna en un *dataset.* Se especifica en la Figura 9.19.

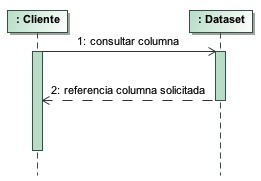


Figura 9.20: Diagrama de secuencia 10: Consultar Columna *Dataset*

### Diagrama de secuencia 11: Intercambiar columnas *Dataset*

El diagrama de secuencia de la Figura 9.20 describe el intercambio entre dos columnas de un *dataset*.

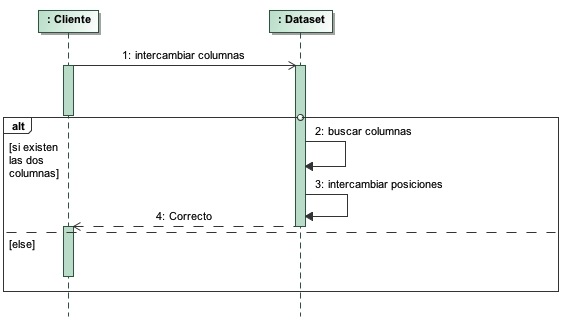


Figura 9.21: Diagrama de secuencia 11: Intercambiar columnas *Dataset*

### Diagrama de secuencia 12: Guardar *Dataset*

Esta operación permite al desarrollador volcar el contenido del *dataset* al sistema de archivos, en cualquier formato definido por la librería. La secuencia de pasos se detalla en la siguiente figura.

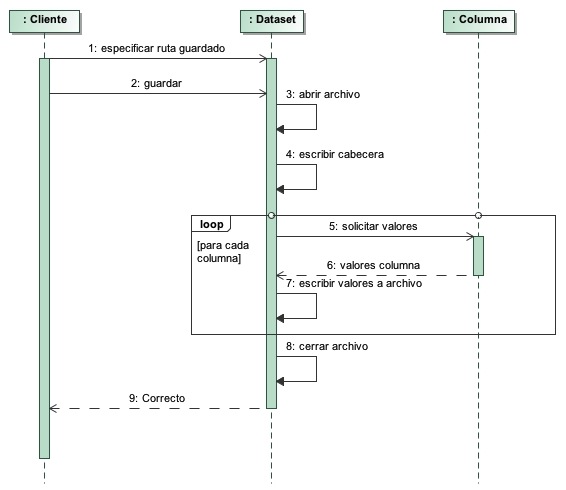


Figura 9.22: Diagrama de secuencia 12. Guardar *Dataset*

### Diagrama de secuencia 13. Ejecutar estrategia

Mediante el seguimiento de los pasos definidos en este diagrama, el desarrollador será capaz de ejecutar un algoritmo de los definidos sobre *dataset*. Su secuencia se especifica en la figura.

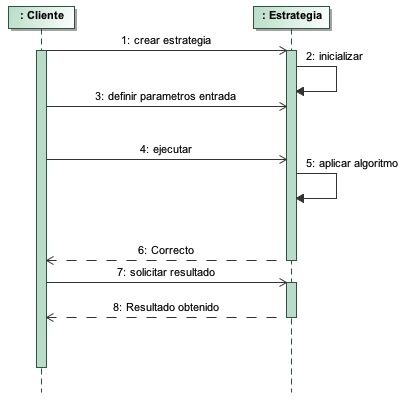


Figura 9.23: Diagrama de secuencia 13. Ejecutar estrategia

### Diagrama de secuencia 14. Crear Proyecto

Mediante este diagrama de secuencia el usuario será capaz de crear un nuevo proyecto en la interfaz de usuario, la secuencia de pasos es la siguiente.

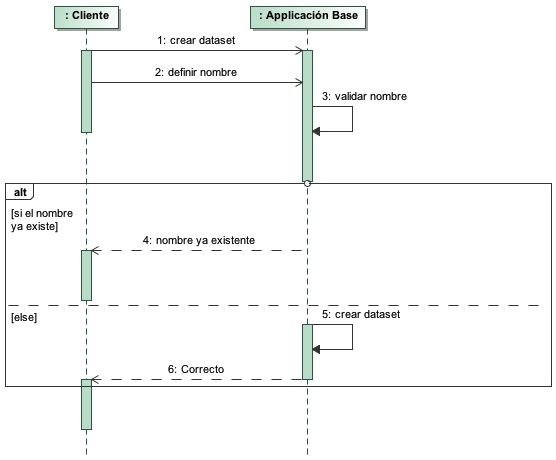


Figura 9.24: Diagrama de secuencia 14. Crear Proyecto

### Diagrama de secuencia 15. Crear *Dataset* Nuevo

Mediante el seguimiento de los pasos descritos en la Figura 9.25, el usuario podrá crear un nuevo *dataset* desde la interfaz usuario.

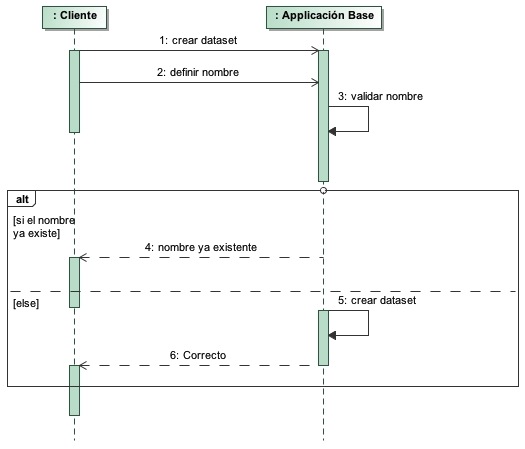


Figura 9.25: Diagrama de secuencia 15. Crear *Dataset* Nuevo

### Diagrama de secuencia 16. Importar *Dataset*

El siguiente diagrama de secuencia detalla los pasos que se deben seguir para que el usuario pueda importar un *dataset* desde el sistema de archivos al entorno de trabajo en la interfaz de usuario.

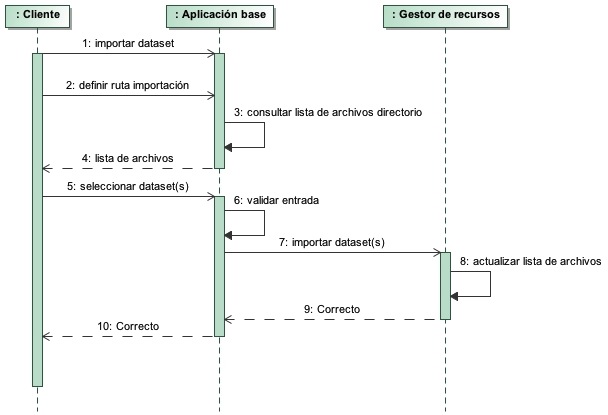


Figura 9.26: Diagrama de secuencia 16. Importar *Dataset*

### Diagrama de secuencia 17. Exportar *Dataset*

Este diagrama, describe cómo un usuario puede exportar un *dataset* desde la interfaz de usuario, se describe en la Figura 9.27.

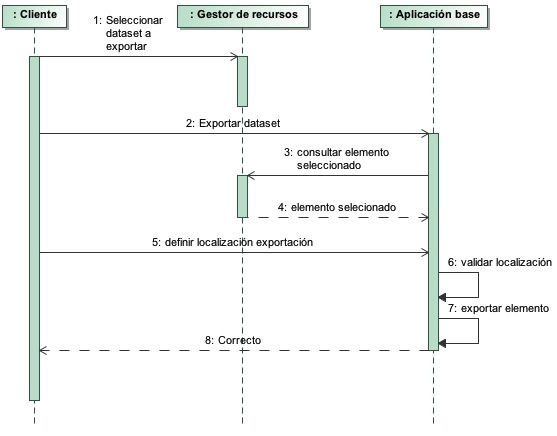


Figura 9.27: Diagrama de secuencia 17. Exportar *Dataset*

### Diagrama de secuencia 18. Cargar *Dataset* en Memoria

El usuario será capaz mediante el diagrama de secuencia descrito en la Figura 9.28, cargar un *dataset* seleccionado en memoria para su posterior manipulación.

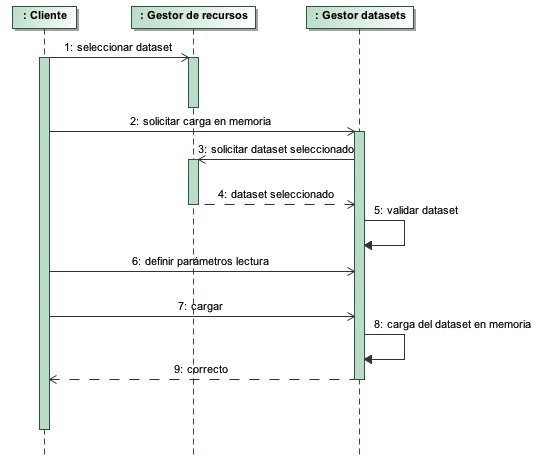


Figura 9.28: Diagrama de secuencia 18. Cargar *Dataset* en Memoria

### Diagrama de secuencia 19. Escribir *Dataset*

Este diagrama de secuencia describe cómo un usuario es capaz de realizar la escritura de un *dataset* previamente cargado. Su especificación se realiza en la Figura 9.29.

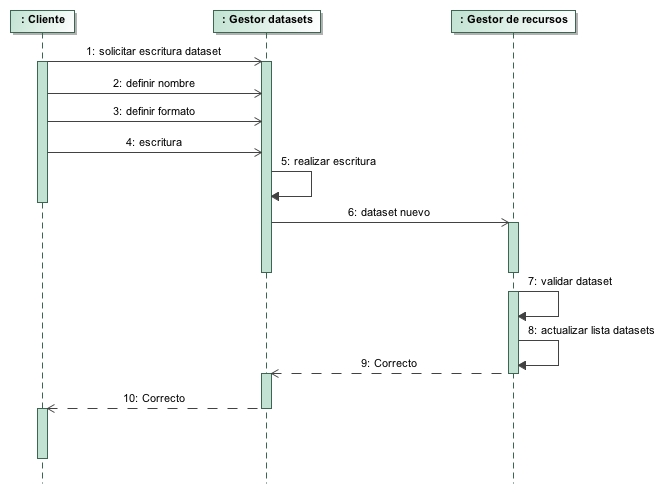


Figura 9.29: Diagrama de secuencia 19. Escribir *Dataset*

### Diagrama de secuencia 20. Aplicar estrategia sobre *Dataset*

El diagrama de secuencia especificado en la Figura 9.30, detalla los pasos a seguir para la aplicación de una estrategia sobre un *dataset* desde la interfaz de usuario.

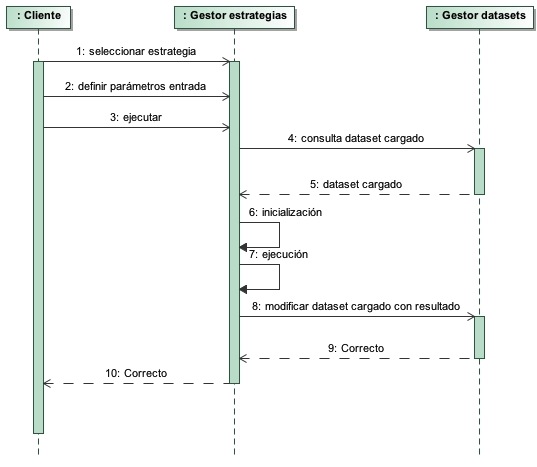


Figura 9.30: Diagrama de secuencia 20. Aplicar estrategia sobre *Dataset*

Parte III

DISEÑO DE LA APLICACIÓN

# DISEÑO PROCEDIMENTAL

En este apartado se realizará un planteamiento inicial acerca de las clases que resultará necesario implementar para el desarrollo de la librería propuesta. Para ello, se utilizará como punto de partida la definición general del problema anteriormente expuesta, así como las explicaciones realizadas durante el proceso de descripción de la información y descripción funcional, incluido en los dos apartados anteriores.

Para cada clase se especificará el componente al que pertenece y la capa arquitectónica, así como las relaciones existentes con las demás del sistema. Estas clases se extraen tanto de los diagramas de caso de uso y los diagramas de secuencia como de la propia definición del problema. Todas se tendrán en cuenta en el modelo de objetos. En cada una de las clases, que se describirán a continuación, se distinguirán principalmente los siguientes apartados:

* Nombre de la clase: nombre que se asignará a la clase para su identificación. Este nombre deberá ser descriptivo, de manera que proporcione una idea acerca del funcionamiento general del elemento al cual pertenece.
* Descripción general de la clase: indicará cual es el objetivo principal de la clase. Se explicará de forma breve el objetivo fundamental de la misma, así como de toda aquella información que pudiera resultar de interés en esta etapa.
* Variables miembro de la clase: en este apartado se analizarán todos aquellos atributos que deberá definir la clase para almacenar la información que ésta necesita para desarrollar la funcionalidad que se le ha asignado de manera correcta. Para cada una de estas variables se indicará su tipo y las principales características a destacar.
* Métodos de la clase: se analizarán todos aquellos métodos definidos por la clase, es decir, aquellas funciones en las que puede estructurarse la funcionalidad desarrollada por la misma.

## Capa *Predata*

Se detallará la capa *Predata* que contiene el núcleo funcional de la librería. En esta sección se detallarán las clases que forman parte de los distintos componentes existentes en la capa. Para mostrarlos se realizará una división por componentes.

### Componente Gestor de Dataset

Este componente se forma por las clases que se ven en la figura a continuación.

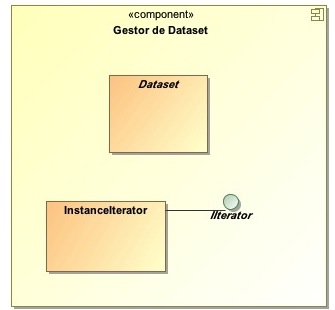


Figura 10.1: Componente Gestor de Dataset

#### Clase Dataset

* Nombre: *Dataset*
* Descripción: esta clase representa la abstracción de un Dataset.
* Variables miembro:
* *sName*: representa el nombre del dataset.
* *iCursor*: representa el índice de la columna que se está usando en un momento determinado.
* *sMissedValue*: el símbolo que representa un valor faltante en el dataset.
* *sNullValue*: el símbolo que representa un valor nulo en el dataset.
* *sEmtpyValue*: el símbolo que representa un valor vacío en el dataset.
* *sCommentedValue*: el símbolo usado para representar un comentario en el dataset.
* *sSeparationValue*: el símbolo usado para representar la separación entre valores de distintas columnas en el dataset.
* *sOpenRangeDelimiter*: el símbolo usado para representar el inicio que determina un valor de rango.
* *sSeparationRangeDelimiter*: el símbolo usado para representar la separación entre los valores mínimo y máximo de un rango.
* *sCloseRangeDelimiter*: el símbolo usado para representar el fin que determina un valor de rango.
* *iNumbersOfDecimals*: variable que almacena el número de decimales que se escribirán en el caso de las columnas de tipo numérico.
* *rgoValidBinaryFalseValues*: almacena la lista de valores falsos, correspondiente a las columnas binarias en el dataset.
* *rgoValidBinaryTrueValues*: almacena la lista de valores verdaderos, correspondiente a las columnas binarias en el dataset.
* Métodos de la clase:
* *Dataset():* constructor sin parámetros de la clase, se encarga inicializar diferentes variables para el correcto uso de la clase.
* *getValidBinaryFalseValues():* método que devuelve la lista de valores válidos falsos, correspondiente a las columnas binarias del *dataset*.
* *getValidBinaryTrueValues():* método que devuelve la lista de valores válidos verdaderos, correspondiente a las columnas binarias del *dataset*.
* *setValidBinaryFalseValues(List):* asigna la lista de valores válidos falsos, correspondiente a las columnas binarias del *dataset*.
* *setValidBinaryTrueValues(List):* asigna la lista de valores válidos verdaderos, correspondiente a las columnas binarias del *dataset*.
* *addColumn(ColumnAbstraction):* método que añade una columna pasada como parámetro al *dataset*.
* *addColumn(ColumnAbstraction, int):* método que añade una columna pasada como parámetro al *dataset*, en la posición indicada como parámetro.
* *removeColumn(int):* método que elimina una columna específica según el índice pasado como parámetro.
* *getColumns():* método que devuelve la lista de columnas que contiene el *dataset*.
* *setColumns(List):* método que asigna la lista de columnas pasada como parámetro al *dataset*.
* *getNumberOfDecimalsToWrite():* método que devuelve el número de decimales que se escribirán para las columna de tipo numérico.
* *setNumberOfDecimalsToWrite():* método que asigna el número de decimales que se escribirán para las columna de tipo numérico.
* *getColumn(int):* método que devuelve la columna situada en el índice pasado como parámetro.
* *getIndexOfColumn(ColumnAbstraction):* método que devuelve el índice de la columna pasada como parámetro.
* *getName():* método que devuelve el nombre del *dataset*.
* *setName(String):* método que asigna el nombre del *dataset* pasado como parámetro.
* *getMissedValue():* método que devuelve la representación en el *dataset* de un valor faltante.
* *setMissedValue(String):* método que asigna la representación en el *dataset* de un valor faltante.
* *getNullValue():* método que devuelve la representación en el *dataset* de un valor nulo.
* *setNullValue(String):* método que asigna la representación en el *dataset* de un valor nulo.
* *getEmtpyValue():* método que devuelve la representación en el *dataset* de un valor vacío.
* *setEmptyValue(String):* método que asigna la representación en el *dataset* de un valor vacío.
* *setRangeDelimiters(String, String, String):* método que asigna los valores delimitadores de apertura, separación y cierre de valores correspondientes a las columnas de tipo rango.
* *getRangeDelimiters():* método que devuelve los valores delimitadores para rangos actualmente asignados.
* *getCommentSymbol():* método que devuelve la representación del símbolo que indica un comentario en el *dataset*.
* *setCommentSymbol(String):* método que asigna la representación del símbolo que indica un comentario en el *dataset*.
* *getSeparationValue():* método que devuelve la representación del símbolo que indica la separación entre dos valores de una misma instancia en el *dataset*.
* *setSeparationValue(String):* método que asigna la representación del símbolo que indica la separación entre dos valores de una misma instancia en el *dataset*.
* *getCursor():* método que devuelve el índice de la columna actualmente manipulada en el *dataset*.
* *setCursor(int):* método que asigna el índice de la columna que actualmente se manipula en el *dataset*.
* *clone():* método que devuelve un *dataset*, con la estructura del *dataset* actual, sin copiar el contenido de las columnas.
* *copy():* método que devuelve una copia integra del *dataset* actual.
* *swapColumns(ColumnAbstraction, ColumnAbstraction):* método que realiza un intercambio entre las dos columnas pasadas como parámetro dentro del *dataset*.
* *open():* método que realiza la apertura de un *dataset*, y lo deja listo para su lectura desde un archivo.
* *close():* método que cierra el archivo asociado al *dataset* en cuestión.
* *reset():* método que realiza un reseteo de todas las variables asociadas a la lectura del *dataset* desde un fichero, para poder leerlo de nuevo.
* *writeDataset(String):* este método realiza la escritura del contenido del *dataset* a un archivo, cuya ruta es pasada como parámetro.
* *readDataset(String, String):* realiza la lectura del *dataset* desde el archivo previamente especificado.
* *addAllValues():* realiza la lectura del contenido de cada una de las columnas del *dataset* desde el fichero.
* *merge(Dataset):* realiza una unión del *dataset* actual con el *dataset* pasado como parámetro.

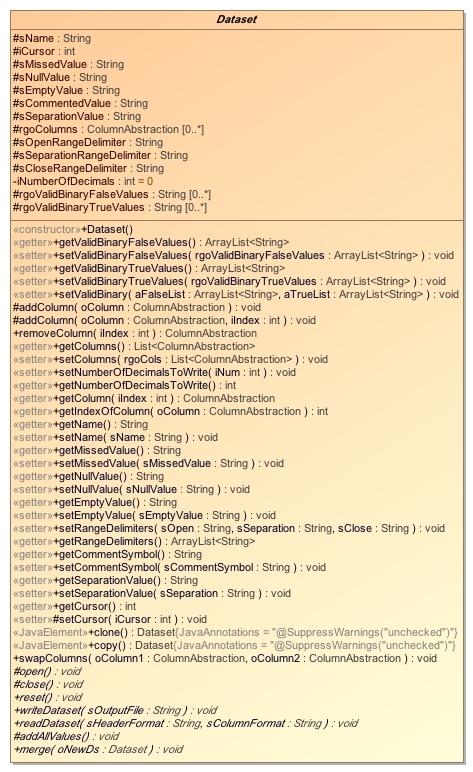


Figura 10.2: Clase *Dataset*

#### Interfaz IIterator

* Nombre: *IIterator*
* Descripción: define la interfaz para el iterador de instancias.
* Métodos de la clase:
* *first():* método que devuelve el primer elemento apuntado por el iterador.
* *next():* método que mueve el iterador al siguiente elemento.
* *isDone():* método que devuelve verdadero si el iterador ya no puede leer más elementos.
* *currentInstance():* método que devuelve la instancia actual a la que apunta el iterador.

### Macintosh HD:Users:psylock:Desktop:IteratorClase.jpg

Figura 10.3: Clase *IIterator*

#### Clase InstanceIterator

* Nombre: *InstanceIterator*
* Descripción: clase que el implementa el iterador de instancias.
* Variables miembro:
* *iRow*: índice de la columna usada en un momento determinado.
* *oData*: *dataset* actualmente usado en el iterador.
* Métodos de la clase:
* *InstanceIterator*(Dataset): constructor parametrizado de la clase, que recibe el *dataset* sobre el que iterar.
* *first():* método que devuelve el primer elemento apuntado por el iterador.
* *next():* método que mueve el iterador al siguiente elemento.
* *isDone():* método que devuelve verdadero si el iterador ya no puede leer más elementos.
* *currentInstance():* método que devuelve la instancia actual a la que apunta el iterador.

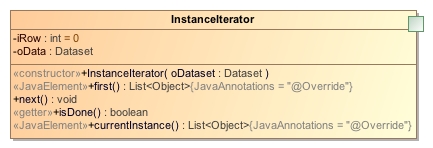


Figura 10.4: Clase *InstanceIterator*

### Componente Gestor de columnas

Este componente está formado por las clases que aparecen en la Figura 10.6. Antes de entrar en la definición de cada uno de los componentes, cabe destacar la presencia de una enumeración llamada *ColumnType (Figura 10.5)*, en el que se listan los tipos de columnas que la librería soporta y que son variables usadas por métodos de este mismo componente.

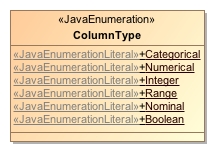


Figura 10.5: Enumeración *ColumnType*

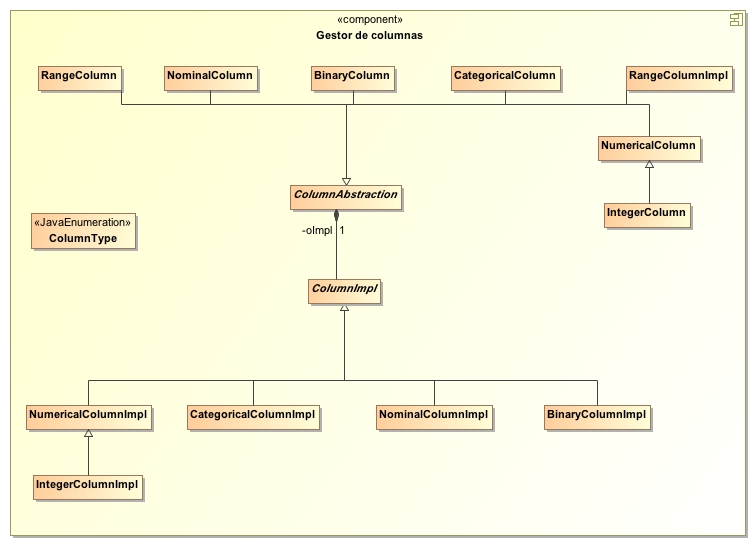


Figura 10.6: Componente Gestor de columnas

#### Clase ColumnAbstraction

* Nombre: *ColumnAbstraction*
* Descripción: es la clase encargada de definir la estructura que deberá tener una columna, en cuanto a metainformación, es desde donde se realizará el acceso a todos los métodos y contenido de la columna, almacenará una variable de tipo *ColumnImpl*, que almacenará el contenido en sí.
* Variables miembro:
* *sName:* variable de tipo *string* que representa el nombre de la columna.
* *enumType:* variable de tipo *ColumnType* que representa el tipo de la columna.
* *oImpl:* variable de tipo *ColumnImpl* que representará la implementación propia de cada columna.
* Métodos de la clase:
* *ColumnAbstraction(String, ColumnType)*: constructor parametrizado de la clase, inicializará el nombre y el tipo pasados como parámetro.
* *getName()*: método que devuelve el nombre de la columna.
* *getType():* método que devuelve el tipo de la columna.
* *addValue(Object):* método que añade un valor pasado como parámetro a la columna.
* *addValue(Object, boolean):* método que fuerza la adición del valor a la columna, si el segundo parámetro es verdadero.
* *addValue(Object, int):* método que añade el valor en el índice de la columna especificado.
* *setValue(Object, int):* método que modifica el valor localizado en el índice pasado como parámetro, por el valor pasado por parámetro.
* *removeValue(int):* método que elimina el valor situado en el índice pasado como parámetro.
* *addAllValues(List<Object>):* método que añade la lista de valores pasada como parámetro a la columna.
* *getElement(int):* método que devuelve el valor situado en el índice pasado como parámetro.
* *getSize():* método que devuelve el número de elementos que contiene la columna.
* *getMissingValue():* método que devuelve el contenido asignado a un valor faltante.
* *setMissingValue(Object):* método que asigna como valor faltante el pasado como parámetro.
* *setMissingValue(Object, boolean):* método que asigna como valor faltante el pasado como parámetro, y además si el segundo parámetro es verdadero, realiza la sustitución en toda la columna del antiguo por el nuevo asignado.
* *getNullValue():* método que devuelve el contenido asignado a un valor nulo.
* *setNullValue(Object):* método que asigna como valor nulo el pasado como parámetro.
* *setNullValue(Object, boolean):* método que asigna como valor nulo el pasado como parámetro, además si el segundo parámetro es verdadero, realiza la sustitución en toda la columna del antiguo por el nuevo asignado.
* *getEmptyValue():* método que devuelve el contenido asignado a un valor vacío.
* *setEmptyValue(Object):* método que asigna como valor vacío el pasado como parámetro.
* *setEmptyValue(Object, boolean):* método que asigna como valor vacío el pasado como parámetro, además si el segundo parámetro es verdadero, realiza la sustitución en toda la columna del antiguo por el nuevo asignado.
* *getValues():* método que devuelve la lista de valores que contiene la columna.

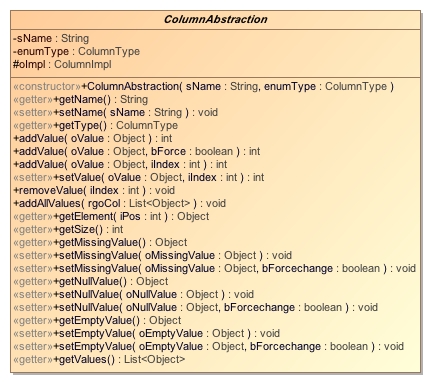


Figura 10.7: Clase *ColumnAbstraction*

#### Clase BinaryColumn

* Nombre: *BinaryColumn*
* Descripción: es la clase encargada de definir una columna de tipo binario. Deriva de *ColumnAbstraction.*
* Métodos de la clase:
* *BinaryColumn():* constructor sin parámetros de la clase, se encarga de inicializar la implementación de la clase y de llamar al constructor de la clase padre.
* *BinaryColumn(String):* constructor de la clase que recibe como parámetro el nombre de la columna.
* *countMissingValues():* llama al método *countMissingValues*() de la implementación.
* *countEmptyValues():* llama al método *countEmptyValues*() de la implementación.
* *countNullValues():* llama al método *countNullValues*() de la implementación.
* *countInvalidValues():* llama al método *countInvalidValues*() de la implementación.
* *toCategorical(String, String):* llama al método *toCategorical*() de la implementación.

### Macintosh HD:Users:psylock:Desktop:Gestor de columnas:Abstraction:BinaryColumn.jpg

Figura 10.8: Clase *BinaryColumn*

#### Clase RangeColumn

* Nombre: *RangeColumn*
* Descripción: es la clase encargada de definir una columna de tipo rango numérico. Deriva de *ColumnAbstraction.*
* Variables miembro:
* *dMinInterval:* variable que almacena el valor mínimo de rango que la columna contiene.
* *dMaxInterval:* variable que almacena el valor máximo de rango que la columna contiene.
* Métodos de la clase:
* *RangeColumn():* constructor sin parámetros de la clase, se encarga de inicializar la implementación de la clase y de llamar al constructor de la clase padre.
* *RangeColumn(String):* constructor de la clase que recibe como parámetro el nombre de la columna.
* *getdMinInterval():* método que devuelve el valor mínimo del rango que la columna contiene.
* *getdMaxInterval():* método que devuelve el valor máximo del rango que la columna contiene.
* *setdMinInterval(double):* método que asigna el valor mínimo del rango que la columna puede contener.
* *setdMaxInterval(double):* método que asigna el valor máximo del rango que la columna puede contener.
* *countMissingValues():* llama al método *countMissingValues*() de la implementación.
* *countNullValues():* llama al método *countNullValues*() de la implementación.
* *countEmptyValues():* llama al método *countEmptyValues*() de la implementación.
* *countInvalidValues():* llama al método *countInvalidValues*() de la implementación.
* *toCategorical():* llama al método *toCategorical*() de la implementación.
* *toNumerical():* llama al método *toNumerical*() de la implementación.

### Macintosh HD:Users:psylock:Desktop:Gestor de columnas:Abstraction:RangeColumn.jpg

Figura 10.9: Clase *RangeColumn*

#### Clase CategoricalColumn

* Nombre: *CategoricalColumn.*
* Descripción: es la clase encargada de definir una columna de tipo categórica. Deriva de *ColumnAbstraction.*
* Métodos de la clase:
* *CategoricalColumn():* constructor sin parámetros de la clase, se encarga de inicializar la implementación de la clase y de llamar al constructor de la clase padre.
* *CategoricalColumn(String):* constructor de la clase que recibe como parámetro el nombre de la columna.
* *setCategoryList(String):* llama al método *setCategoryList*() de la implementación.
* *getElementIndex(int):* llama al método *getElementIndex()* de la implementación.
* *getCategoryList():* llama al método *getCategoryList()* de la implementación.
* *getSortedCategoryList():* llama al método *getSortedCategoryList()* de la implementación.
* *getCategoryIndex(String):* llama al método *getCategoryIndex()* de la implementación.
* *countMissingValues():* llama al método *countMissingValues()* de la implementación.
* *countEmptyValues():* llama al método *countEmptyValues()* de la implementación.
* *countNullValues():* llama al método *countNullValues()* de la implementación.
* *countInvalidValues():* llama al método *countInvalidValues()* de la implementación.
* *toNominal():* llama al método *toNominal()* de la implementación.
* *toBinary(List<String>):* llama al método *toBinary()* de la implementación.
* *toNumerical():* llama al método *toNumerical()* de la implementación.
* *replaceCategory(String, String, boolean):* llama al método *replaceCategory()* de la implementación.

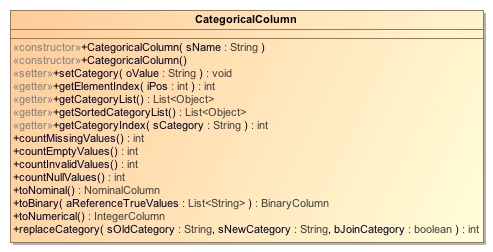
**

Figura 10.10: Clase *CategoricalColumn*

#### Clase NominalColumn

* Nombre: *NominalColumn*
* Descripción: es la clase encargada de definir una columna de tipo nominal. Deriva de *ColumnAbstraction.*
* Métodos de la clase:
* *NominalColumn():* constructor sin parámetros de la clase, se encarga de inicializar la implementación de la clase y de llamar al constructor de la clase padre.
* *NominalColumn(String):* constructor de la clase que recibe como parámetro el nombre de la columna.
* *countMissingValues():* llama al método *countMissingValues()* de la implementación.
* *countEmptyValues():* llama al método *countEmptyValues()* de la implementación.
* *countNullValues():* llama al método *countNullValues()* de la implementación.
* *countInvalidValues():* llama al método *countInvalidValues()* de la implementación.
* *toCategorical():* llama al método toCategorical() de la implementación.
* *toNumerical():* llama al método toNumerical() de la implementación.

### Macintosh HD:Users:psylock:Desktop:Gestor de columnas:Abstraction:NominalColumn.jpg

Figura 10.11: Clase *NominalColumn*

#### Clase NumericalColumn

* Nombre: *NumericalColumn*
* Descripción: es la clase encargada de definir una columna de tipo numérico (números reales). Deriva de *ColumnAbstraction.*
* Variables miembro:
* *dMinInterval*: representa el valor mínimo que almacena la columna.
* *dMaxInterval*: representa el valor máximo que almacena la columna.
* Métodos de la clase:
* *NumericalColumn():* constructor sin parámetros de la clase, se encarga de inicializar la implementación de la clase y de llamar al constructor de la clase padre.
* *NumericalColumn(String, ColumnType):* constructor de la clase que recibe como parámetro el nombre de la columna y el tipo de columna.
* *getdMinInterval():* método que devuelve el mínimo valor que almacena la columna.
* *setdMinInterval(double):* método que establece el contenido del mínimo valor que almacena la columna.
* *getdMaxInterval():* método que devuelve el máximo valor que almacena la columna.
* *setdMaxInterval(double):* método que establece el contenido del máximo valor que almacena la columna.
* *getValues():* método que devuelve la lista de valores que la columna contiene.
* *countMissingValues():* llama al método *countMissingValues*() de la implementación.
* *countEmptyValues():* llama al método *countEmptyValues*() de la implementación.
* *countNullValues():* llama al método *countNullValues*() de la implementación.
* *countInvalidValues():* llama al método *countInvalidValues*() de la implementación.
* *getMaxValue():* llama al método *getMaxValue()* de la implementación.
* *getMinValue():* llama al método *getMinValue()* de la implementación.
* *mean():* llama al método *mean()* de la implementación.
* *normalize():* llama al método *normalize()* de la implementación.
* *standarize(double, double):* llama al método *standarize()* de la implementación.
* *discretizeByEqualWidth(int):* llama al método *discretizeByEqualWidth()* de la implementación.
* *discretizeByEqualFrequency(int):* llama al método *discretizeByEqualFrequency()* de la implementación.
* *toNominal():* llama al método *toNominal()* de la implementación.
* *toInteger(boolean):* llama al método *toInteger()* de la implementación.
* *discretizeBySize(int):* llama al método *discretizeBySize()* de la implementación.
* *discretizeByAmplitude(double):* llama al método *discretizeByAmplitude()* de la implementación.

### Macintosh HD:Users:psylock:Desktop:Gestor de columnas:Abstraction:NumericalColumn.jpg

Figura 10.12: Clase *NumericalColumn*

#### Clase IntegerColumn

* Nombre: *IntegerColumn*
* Descripción: se trata de la clase encargada de definir una columna de tipo entero, es decir, números sin parte decimal.
* Métodos de la clase:
* *IntegerColumn():* constructor sin parámetros de la clase, se encarga de inicializar la implementación de la clase y de llamar al constructor de la clase padre.
* *IntegerColumn(String):* constructor de la clase que recibe como parámetro el nombre de la columna.
* *getMaxValue():* llama al método *getMaxValue()* de la implementación.
* *getMinValue():* llama al método *getMinValue()* de la implementación.
* *getValues():* método que devuelve la lista de valores que almacena la columna.
* *discretizeByEqualWidth(int):* llama al método *discretizeByEqualWidth()* de la implementación.
* *discretizeByEqualFrequency(int):* llama al método *discretizeByEqualFrequency()* de la implementación.
* *discretizeBySize(int):* llama al método *discretizeBySize()* de la implementación.
* *discretizeByAmplitude(double):* llama al método *discretizeByAmplitude()* de la implementación.
* *toCategorical():* llama al método toCategorical() de la implementación.

### Macintosh HD:Users:psylock:Desktop:Gestor de columnas:Abstraction:IntegerColumn.jpg

Figura 10.13: Clase *IntegerColumn*

#### Clase ColumnImpl

* Nombre: *ColumnImpl*
* Descripción: esta será la clase encargada de definir la implementación para una columna, de esta derivará cada una de las implementaciones para cada columna.
* Variables miembro:
* *oMissingValue:* representa un valor faltante.
* *oNullValue:* representa un valor nulo.
* *oEmptyValue:* representa un valor vacío.
* Métodos de la clase:
* *getMissingValue():* método que devuelve la representación asignada de un valor faltante.
* *setMissingValue(Object):* método que asigna la representación de un valor faltante.
* *setMissingValue(Object, boolean):* método que asigna la representación asignada de un valor faltante, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *getNullValue():* método que devuelve la representación de un valor nulo.
* *setNullValue():* método que asigna la representación de un valor nulo.
* *setNullValue(Object, boolean):* método que asigna la representación de un valor nulo, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *getEmptyValue():* método que devuelve la representación asignada de un valor vacío.
* *setEmptyValue():* método que asigna la representación de un valor vacío.
* *setEmptyValue(Object, boolean):* método que asigna la representación de un valor vacío, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *addValue(Object):* método que añade el valor pasado como parámetro a la columna.
* *addValue(Object, boolean):* método que fuerza la adición del valor pasado como parámetro a la columna si el segundo parámetro es verdadero.
* *addValue(Object, int):* método que añade el valor pasado como parámetro a la columna en el índice especificado como parámetro.
* *setValue(Object, int):* método que modifica el valor situado en el índice pasado como parámetro por el valor pasado como parámetro.
* *removeValue(int):* método que elimina el valor situado en el índice especificado como parámetro.
* *addAllValues(List):* método que añade a la columna, la lista de valores pasada como parámetro.
* *getValues():* método que devuelve la lista de valores que contiene la columna.
* *getElement(int):* método que devuelve el valor situado en el índice pasado como parámetro de la columna.
* *getSize():* método que devuelve el número de valores actuales que la columna contiene.

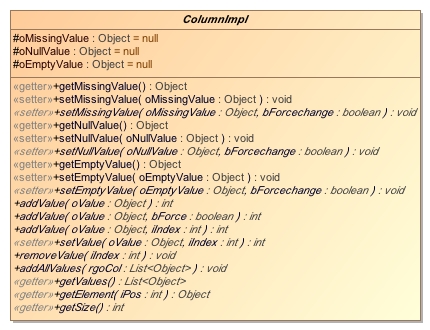


Figura 10.14: Clase *ColumnImpl*

#### Clase BinaryColumnImpl

* Nombre: *BinaryColumnImpl*
* Descripción: esta clase representa la implementación de una columna de tipo binario.
* Variables miembro:
* *rgoValues:* representa la lista de valores que almacena la columna.
* Métodos de la clase:
* *BinaryColumnImpl():* constructor sin parámetros de la clase, se encarga de inicializar los valores nulos, vacíos y faltantes.
* *addValue(Object):* método que añade el valor pasado como parámetro a la columna.
* *addValue(Object, boolean):* método que fuerza la adición del valor pasado como parámetro a la columna si el segundo parámetro es verdadero.
* *addValue(Object, int):* método que añade el valor pasado como parámetro a la columna en el índice especificado como parámetro.
* *setValue(Object, int):* método que modifica el valor situado en el índice pasado como parámetro por el valor pasado como parámetro.
* *removeValue(int):* método que elimina el valor situado en el índice especificado como parámetro.
* *addAllValues(List):* método que añade a la columna, la lista de valores pasada como parámetro.
* *getValues():* método que devuelve la lista de valores que contiene la columna.
* *getElement(int):* método que devuelve el valor situado en el índice pasado como parámetro de la columna.
* *getSize():* método que devuelve el número de valores actuales que la columna contiene.
* *countMissingValues():* devuelve el número de valores faltantes que contiene la columna.
* *countEmptyValues():* devuelve el número de valores vacíos que contiene la columna.
* *countNullValues():* devuelve el número de valores nulos que contiene la columna.
* *countInvalidValues():* devuelve el número de valores inválidos que contiene la columna.
* *setMissingValue(Object, boolean):* método que asigna la representación asignada de un valor faltante, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setNullValue(Object, boolean):* método que asigna la representación de un valor nulo, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setEmptyValue(Object, boolean):* método que asigna la representación de un valor vacío, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *toCategorical(String, String, String):* método que devuelve la transformación de la columna binaria en categórica.

### Macintosh HD:Users:psylock:Desktop:Gestor de columnas:Implementation:BinaryColumnImpl.jpg

Figura 10.15: Clase *BinaryColumnImpl*

#### Clase RangeColumnImpl

* Nombre: *RangeColumnImpl*
* Descripción: esta clase representa la implementación de una columna de tipo rango.
* Variables miembro:
* *rgoMinIntervalValues:* representa la lista de los valores mínimos de cada rango almacenado en la columna.
* *rgoMaxIntervalValues:* representa la lista de los valores máximos de cada rango almacenado en la columna.
* Métodos de la clase:
* *RangeColumnImpl():* constructor sin parámetros de la clase, se encarga de inicializar los valores nulos, vacíos y faltantes.
* *addValue(Object):* método que añade el valor pasado como parámetro a la columna.
* *addValue(Object, boolean):* método que fuerza la adición del valor pasado como parámetro a la columna si el segundo parámetro es verdadero.
* *addValue(Object, int):* método que añade el valor pasado como parámetro a la columna en el índice especificado como parámetro.
* *setValue(Object, int):* método que modifica el valor situado en el índice pasado como parámetro por el valor pasado como parámetro.
* *removeValue(int):* método que elimina el valor situado en el índice especificado como parámetro.
* *addAllValues(List):* método que añade a la columna, la lista de valores pasada como parámetro.
* *getValues():* método que devuelve la lista de valores que contiene la columna.
* *getElement(int):* método que devuelve el valor situado en el índice pasado como parámetro de la columna.
* *getSize():* método que devuelve el número de valores actuales que la columna contiene.
* *countMissingValues():* devuelve el número de valores faltantes que contiene la columna.
* *countEmptyValues():* devuelve el número de valores vacíos que contiene la columna.
* *countNullValues():* devuelve el número de valores nulos que contiene la columna.
* *countInvalidValues():* devuelve el número de valores inválidos que contiene la columna.
* *setMissingValue(Object, boolean):* método que asigna la representación asignada de un valor faltante, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setNullValue(Object, boolean):* método que asigna la representación de un valor nulo, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setEmptyValue(Object, boolean):* método que asigna la representación de un valor vacío, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* toCategorical(String): método que devuelve la transformación de la columna de rango en categórica.
* toNumerical(String, int): método que devuelve la transformación de la columna de rango en numérica.

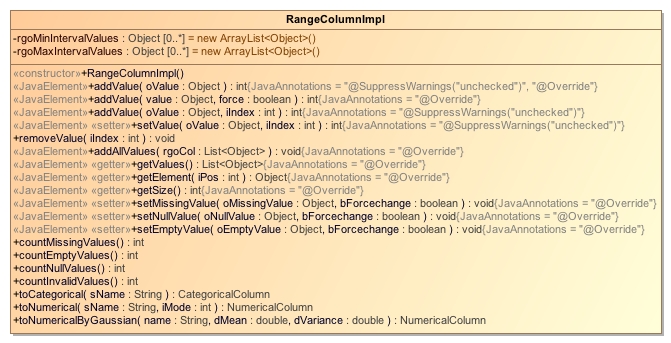


Figura 10.16: Clase *RangeColumnImpl*

#### Clase CategoricalColumnImpl

* Nombre: *CategoricalColumnImpl*
* Descripción: esta clase representa la implementación de una columna de tipo categórico.
* Variables miembro
* *rgoValues*: representa la lista de valores que almacena la columna.
* *hCategoryTable*: representa la correspondencia entre las categorías representadas como cadenas de caracteres y el valor que en realidad se almacena en la columna (enteros).
* Métodos de la clase:
* *CategoricalColumnImpl():* constructor sin parámetros de la clase, se encarga de inicializar los valores nulos, vacíos y faltantes.
* *addValue(Object):* método que añade el valor pasado como parámetro a la columna.
* *addValue(Object, boolean):* método que fuerza la adición del valor pasado como parámetro a la columna si el segundo parámetro es verdadero.
* *addValue(Object, int):* método que añade el valor pasado como parámetro a la columna en el índice especificado como parámetro.
* *setValue(Object, int):* método que modifica el valor situado en el índice pasado como parámetro por el valor pasado como parámetro.
* *removeValue(int):* método que elimina el valor situado en el índice especificado como parámetro.
* *addAllValues(List):* método que añade a la columna, la lista de valores pasada como parámetro.
* *getValues():* método que devuelve la lista de valores que contiene la columna.
* *getElement(int):* método que devuelve el valor situado en el índice pasado como parámetro de la columna.
* *getSize():* método que devuelve el número de valores actuales que la columna contiene.
* *countMissingValues():* devuelve el número de valores faltantes que contiene la columna.
* *countEmptyValues():* devuelve el número de valores vacíos que contiene la columna.
* *countNullValues():* devuelve el número de valores nulos que contiene la columna.
* *countInvalidValues():* devuelve el número de valores inválidos que contiene la columna.
* *setMissingValue(Object, boolean):* método que asigna la representación asignada de un valor faltante, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setNullValue(Object, boolean):* método que asigna la representación de un valor nulo, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setEmptyValue(Object, boolean):* método que asigna la representación de un valor vacío, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *getElementIndex(int):* método que devuelve el índice de la categoría almacenada en la posición pasada como parámetro.
* *setCategory(String):* método que añade la categoría pasada como parámetro, como categoría válida en la columna.
* *getCategoryList():* método que devuelve la lista de categorías posibles en la columna.
* *getSortedCategoryList():* método que devuelve la lista de categorías posibles en la columna, ordenadas según se han ido añadiendo.
* *getCategoryIndex(String):* método que devuelve el índice asociado a la categoría pasada como parámetro.
* *toNominal(String):* método que devuelve la transformación de la columna categórica en nominal.
* *toBinary(List, String):* método que devuelve la transformación de la columna categórica en binaria.
* *toNumerical(String):* método que devuelve la transformación de la columna categórica en numérica, en este caso, los valores numéricos se corresponden con los índices asociados a cada categoría.
* *replaceCategory(String, String, boolean):* método que realiza un reemplazo entre las categorías pasadas como parámetro, en toda la columna.

### Macintosh HD:Users:psylock:Desktop:Gestor de columnas:Implementation:CategoricalColumnImpl.jpg

Figura 10.17: Clase *CategoricalColumnImpl*

#### Clase NominalColumnImpl

* Nombre: *NominalColumnImpl*
* Descripción: esta clase representa la implementación de una columna de tipo nominal.
* Variables miembro:
* *rgoValues:* representa la lista de valores que almacena la columna.
* Métodos de la clase:
* *NominalColumnImpl():* constructor sin parámetros de la clase, se encarga de inicializar los valores nulos, vacíos y faltantes.
* *addValue(Object):* método que añade el valor pasado como parámetro a la columna.
* *addValue(Object, boolean):* método que fuerza la adición del valor pasado como parámetro a la columna si el segundo parámetro es verdadero.
* *addValue(Object, int):* método que añade el valor pasado como parámetro a la columna en el índice especificado como parámetro.
* *setValue(Object, int):* método que modifica el valor situado en el índice pasado como parámetro por el valor pasado como parámetro.
* *removeValue(int):* método que elimina el valor situado en el índice especificado como parámetro.
* *addAllValues(List):* método que añade a la columna, la lista de valores pasada como parámetro.
* *getValues():* método que devuelve la lista de valores que contiene la columna.
* *getElement(int):* método que devuelve el valor situado en el índice pasado como parámetro de la columna
* *getSize():* método que devuelve el número de valores actuales que la columna contiene.
* *countMissingValues():* devuelve el número de valores faltantes que contiene la columna.
* *countEmptyValues():* devuelve el número de valores vacíos que contiene la columna.
* *countNullValues():* devuelve el número de valores nulos que contiene la columna.
* *countInvalidValues():* devuelve el número de valores inválidos que contiene la columna.
* *setMissingValue(Object, boolean):* método que asigna la representación asignada de un valor faltante, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setNullValue(Object, boolean):* método que asigna la representación de un valor nulo, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setEmptyValue(Object, boolean):* método que asigna la representación de un valor vacío, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *toCategorical(String):* método que devuelve la transformación de la columna nominal en categórica.
* *toNumerical(String):* método que devuelve la transformación de la columna nominal en numérica.

### Macintosh HD:Users:psylock:Desktop:Gestor de columnas:Implementation:NominalColumnImpl.jpg

Figura 10.18: Clase *NominalColumnImpl*

#### Clase NumericalColumnImpl

* Nombre: *NumericalColumnImpl*
* Descripción: esta clase representa la implementación de una columna de tipo numérico real.
* Variables miembro:
* rgoValues: representa la lista de valores que almacena la columna.
* oComparatorAscendent: variable que es usada para poder ordenar los valores numéricos reales de la columna de manera ascendente, tratando los valores nulos, vacíos y faltantes, como menores que cualquier otro valor.
* Métodos de la clase:
* *NumericalColumnImpl():* constructor sin parámetros de la clase, se encarga de inicializar los valores nulos, vacíos y faltantes.
* *addValue(Object):* método que añade el valor pasado como parámetro a la columna.
* *addValue(Object, boolean):* método que fuerza la adición del valor pasado como parámetro a la columna si el segundo parámetro es verdadero.
* *addValue(Object, int):* método que añade el valor pasado como parámetro a la columna en el índice especificado como parámetro.
* *setValue(Object, int):* método que modifica el valor situado en el índice pasado como parámetro por el valor pasado como parámetro.
* *removeValue(int):* método que elimina el valor situado en el índice especificado como parámetro.
* *addAllValues(List):* método que añade a la columna, la lista de valores pasada como parámetro.
* *getValues():* método que devuelve la lista de valores que contiene la columna.
* *getElement(int):* método que devuelve el valor situado en el índice pasado como parámetro de la columna.
* *getSize():* método que devuelve el número de valores actuales que la columna contiene.
* *countMissingValues():* devuelve el número de valores faltantes que contiene la columna.
* *countEmptyValues():* devuelve el número de valores vacíos que contiene la columna.
* *countNullValues():* devuelve el número de valores nulos que contiene la columna.
* *countInvalidValues():* devuelve el número de valores inválidos que contiene la columna.
* *setMissingValue(Object, boolean):* método que asigna la representación asignada de un valor faltante, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setNullValue(Object, boolean):* método que asigna la representación de un valor nulo, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *setEmptyValue(Object, boolean):* método que asigna la representación de un valor vacío, y además, si el segundo parámetro es verdadero, realiza la sustitución de la representación antigua por la nueva.
* *discretizeByEqualWidth(int, String):* método que realiza una discretización por igual anchura de la columna numérica, devolviendo el resultado en una columna de tipo rango.
* *discretizeByEqualFrequency(int, String):* método que realiza una discretización por igual frecuencia de la columna numérica, devolviendo el resultado en una columna de tipo rango.
* *discretizeBySize(int, String):* método que realiza una discretización por igual tamaño de partición de la columna numérica, devolviendo el resultado en una columna de tipo rango.
* *discretizeByAmplitude(double, String):* método que realiza una discretización por igual amplitud de partición de la columna numérica, devolviendo el resultado en una columna de tipo rango.
* *toNominal(String):* método que devuelve la transformación de la columna numérica en nominal.
* *toInteger(String, boolean):* método que devuelve la transformación de la columna numérica en una columna de tipo numérica entera.
* *mean():* método que devuelve la media aritmética de los valores de la columna.
* *getMaxValue():* método que devuelve el máximo valor numérico que contiene la columna.
* *getMinValue():* método que devuelve el mínimo valor numérico que contiene la columna.
* *normalize():* método que realiza una normalización N(0,1) de los valores de la columna.
* *standarize(double, double):* método que realiza una estandarización dada una media y varianza pasadas como parámetro, de los valores de la columna.

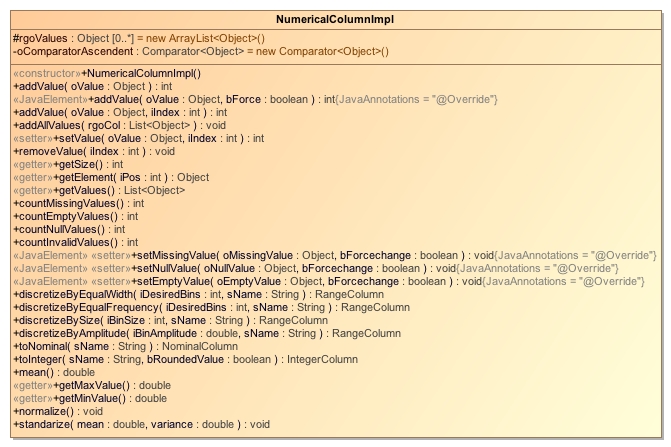


Figura 10.19: Clase *NumericalColumnImpl*

#### Clase IntegerColumnImpl

* Nombre: *IntegerColumnImpl*
* Descripción: esta clase representa la implementación de una columna de tipo numérico entero sin decimales.
* Variables miembro:
* oComparatorAscendent: variable que es usada para poder ordenar los valores numéricos reales de la columna de manera ascendente, tratando los valores nulos, vacíos y faltantes, como menores que cualquier otro valor.
* Métodos de la clase:
* *IntegerColumnImpl():* constructor sin parámetros de la clase, se encarga de inicializar los valores nulos, vacíos y faltantes.
* *addValue(Object):* método que añade el valor pasado como parámetro a la columna.
* *addValue(Object, boolean):* método que fuerza la adición del valor pasado como parámetro a la columna si el segundo parámetro es verdadero.
* *addValue(Object, int):* método que añade el valor pasado como parámetro a la columna en el índice especificado como parámetro.
* *setValue(Object, int):* método que modifica el valor situado en el índice pasado como parámetro por el valor pasado como parámetro.
* *getMaxValue():* método que devuelve el máximo valor numérico que contiene la columna.
* *getMinValue():* método que devuelve el mínimo valor numérico que contiene la columna.
* *discretizeByEqualWidth(int, String):* método que realiza una discretización por igual anchura de la columna numérica, devolviendo el resultado en una columna de tipo rango.
* *discretizeByEqualFrequency(int, String):* método que realiza una discretización por igual frecuencia de la columna numérica, devolviendo el resultado en una columna de tipo rango.
* *discretizeBySize(int, String):* método que realiza una discretización por igual tamaño de partición de la columna numérica, devolviendo el resultado en una columna de tipo rango.
* *discretizeByAmplitude(double, String):* método que realiza una discretización por igual amplitud de partición de la columna numérica, devolviendo el resultado en una columna de tipo rango.
* *toCategorical(String):* método que devuelve la transformación de la columna de tipo entero en una columna de tipo categórico.

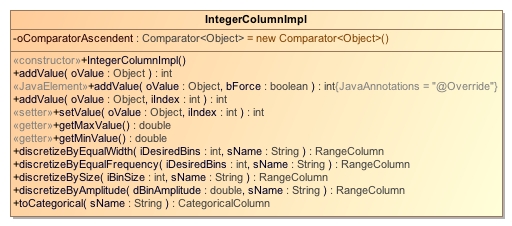


Figura 10.20: Clase *IntegerColumnImpl*

### Componente Serializador

Este componente es el encargado de la gestión de entrada y salida de los *datasets*, básicamente proporciona los métodos para obtener y volcar *datasets* desde/hacia el sistema de archivos, su estructura se define en la Figura 10.21.

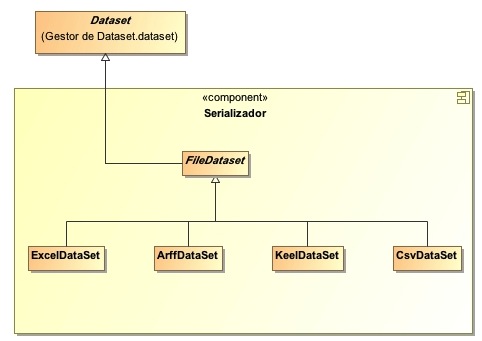


Figura 10.21: Componente Serializador

#### Clase FileDataset

* Nombre: *FileDataset*
* Descripción: esta clase representa un *dataset* que se obtiene por medio un archivo situado en el sistema de archivos, de cualquier formato.
* Variables miembro:
* *sFileName:* representa la ruta donde se sitúa el *dataset* en el sistema de archivos.
* *oFileReader:* representa el flujo de entrada del archivo.
* Métodos de la clase:
* *FileDataset():* constructor sin parámetros de la clase que se encarga de llamar al constructor de la clase padre, en este caso de la clase *Dataset.*
* *FileDataset(String):* constructor parametrizado, que recibe como parámetro la ruta donde se encuentra el *dataset* dentro del sistema de archivos.
* *getFileName():* método que devuelve la ruta donde se encuentra el *dataset* en el sistema de archivos.
* *setFileName(String):* método que asigna la ruta donde se encuentra el *dataset* dentro del sistema de archivos.

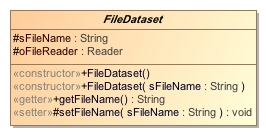


Figura 10.22: Clase *FileDataset*

#### Clase KeelDataSet

* Nombre: *KeelDataSet*
* Descripción: esta clase representa un *dataset* que se lee desde el sistema de archivos, cabe destacar que deriva de la clase *FileDataset* y a su vez de la clase *Dataset*, en este caso, el dataset es del formato KEEL detallado en el Capítulo 8.
* Variables miembro:
* *KEEL\_RELATION*: variable que almacena la palabra clave que denota el nombre del dataset.
* *KEEL\_ATTRIBUTE:* variable que almacena la palabra clave que denota la definición de un atributo del dataset.
* *KEEL\_DATA:* variable que almacena la palabra clave que denota el inicio de los datos en sí del *dataset*.
* *KEEL\_OUTPUT:* variable que almacena la palabra clave que denota un atributo de salida.
* *KEEL\_INPUT*: variable que almacena la palabra clave que denota un atributo de entrada.
* *sBufferInstance*: variable que sirve como buffer para poder leer las instancias del dataset.
* Métodos de la clase:
* *KeelDataSet()*: constructor sin parámetros de la clase.
* *KeelDataSet(String):* constructor parametrizado de la clase, que recibe la ruta del archivo a cargar.
* *open():* método que se encarga de obtener la cabecera del *dataset.*
* *close():* método que cierra el archivo del *dataset*.
* *obtainMetadata(String):* método que se encarga de realizar la lectura de la cabecera del *dataset*.
* *addAttributeToSpecification(String, String, String):* método que dada una especificación de un atributo en la cabecera del dataset, añade la columna vacía al *dataset,* para su posterior lectura de contenido.
* *readDataset(String, String):* método que se encarga del proceso completo de lectura del *dataset*.
* *writeDataset(String):* método que realiza la escritura del contenido del dataset a un archivo en la ruta especificada como parámetro.
* *addAllValues():* método que realiza la lectura de los datos del *dataset*, y los añade a las columnas correspondientes en cada caso.
* *merge(Dataset):* método que realiza la unión del *dataset* pasado como parámetro al actual.

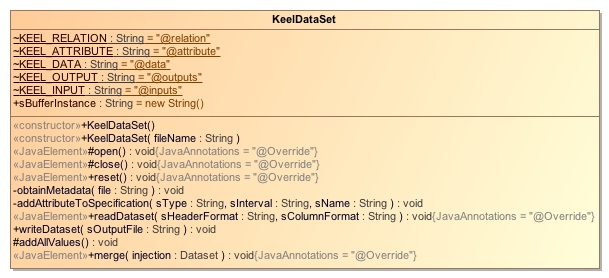


Figura 10.23: Clase *KeelDataSet*

#### Clase ArffDataSet

* Nombre: *ArffDataSet*
* Descripción: esta clase representa un *dataset* que se lee desde el sistema de archivos, cabe destacar que deriva de la clase *FileDataset* y a su vez de la clase *Dataset*, en este caso, el dataset es del formato Weka anteriormente detallado.
* Variables miembro:
* *ARFF\_RELATION*: variable que almacena la palabra clave que denota el nombre del dataset.
* *ARFF\_ATTRIBUTE:* variable que almacena la palabra clave que denota la definición de un atributo del dataset.
* *ARFF\_DATA:* variable que almacena la palabra clave que denota el inicio de los datos en sí del *dataset*.
* *ARFF\_OUTPUT:* variable que almacena la palabra clave que denota un atributo de salida.
* *ARFF\_INPUT*: variable que almacena la palabra clave que denota un atributo de entrada.
* *sBufferInstance*: variable que sirve como buffer para poder leer las instancias del dataset.
* Métodos de la clase:
* *ArffDataSet()*: constructor sin parámetros de la clase.
* *ArffDataSet (String):* constructor parametrizado de la clase, que recibe la ruta del archivo a cargar.
* *open():* método que se encarga de obtener la cabecera del *dataset.*
* *close():* método que cierra el archivo del *dataset*.
* *reset():* método que realiza un reseteo de todas las variables asociadas a la lectura del *dataset* desde un fichero, para poder leerlo de nuevo.
* *obtainMetadata(String):* método que se encarga de realizar la lectura de la cabecera del *dataset*.
* *addAttributeToSpecification(String, String, String):* método que dada una especificación de un atributo en la cabecera del dataset, añade la columna vacía al *dataset,* para su posterior lectura de contenido.
* *readDataset(String, String):* método que se encarga del proceso completo de lectura del *dataset*.
* *writeDataset(String):* método que realiza la escritura del contenido del dataset a un archivo en la ruta especificada como parámetro.
* *addAllValues():* método que realiza la lectura de los datos del *dataset*, y los añade a las columnas correspondientes en cada caso.
* *merge(Dataset):* método que realiza la unión del *dataset* pasado como parámetro al actual.

### Macintosh HD:Users:psylock:Desktop:ArffDatasetClase.jpg

Figura 10.24: Clase *ArffDataSet*

#### Clase CsvDataSet

* Nombre: *CsvDataSet*
* Descripción: esta clase representa un *dataset* que se lee desde el sistema de archivos, cabe destacar que deriva de la clase *FileDataset* y a su vez de la clase *Dataset*, en este caso, el dataset es del formato *csv* anteriormente detallado.
* Variables miembro:
* *bReadNamesAtStart:* si es verdadero, los nombres de las columnas serán leídos antes que las columnas.
* *iSkipStartLines:* variable que almacena el número de líneas del *dataset* al inicio que se desecharan en la lectura.
* *sInputColumnFormat:* variable usada para especificar los tipos de columna que contiene el *dataset*.
* *iNumFormatColumns:* número de columnas especificadas en el formato de columnas de la variable anterior.
* *qBufferLine*: se trata de una cola, usada como buffer en la lectura de instancias del dataset.
* *bNeedReadNamesAtEnd*: si es verdadero, los nombres serán leídos al final del fichero.
* *iSkipEndLines*: número de líneas que se desechan de la lectura al final del fichero.
* *iSkipEndNameLines*: número de líneas que se desechan de la lectura después de leer los nombres de las columnas.
* *bOnlyReadNames*: si es verdadero, solo se leerán los nombres de las columnas.
* *sBufferInstance*: variable que sirve como buffer para poder leer las instancias del dataset.
* Métodos de la clase:
* *CsvDataSet()*: constructor sin parámetros de la clase.
* *CsvDataSet(String):* constructor parametrizado de la clase, que recibe la ruta del archivo a cargar.
* *open():* método que se encarga de obtener la cabecera del *dataset.*
* *close():* método que cierra el archivo del *dataset*.
* *reset():* método que realiza un reseteo de todas las variables asociadas a la lectura del *dataset* desde un fichero, para poder leerlo de nuevo.
* *obtainMetadata(String):* método que se encarga de realizar la lectura de la cabecera del *dataset*.
* *readDataset(String, String):* método que se encarga del proceso completo de lectura del *dataset*.
* *writeDataset(String):* método que realiza la escritura del contenido del dataset a un archivo en la ruta especificada como parámetro.
* *addAllValues():* método que realiza la lectura de los datos del *dataset*, y los añade a las columnas correspondientes en cada caso.
* *merge(Dataset):* método que realiza la unión del *dataset* pasado como parámetro al actual.
* *setFormatHeaderSpecification():* método que dada una especificación de los atributos en la variable *sInputColumnFormat*, añade todas las columnas vacías según el tipo especificado en la variable.
* *getOperationsArrayFromInput(String):* método que devuelve un array, donde cada elemento es una palabra clave de lo que hay que leer en cada momento, sirve para controlar el orden de lectura de los elementos del *dataset* (nombres, instancias o saltos de línea).
* *readNames(int):* método que realiza la lectura de los nombres del *dataset.*

### Macintosh HD:Users:psylock:Desktop:CSVDatasetClase.jpg

Figura 10.25: Clase CsvDataSet

#### Clase ExcelDataSet

* Nombre: *ExcelDataSet*
* Descripción: esta clase representa un *dataset* que se lee desde el sistema de archivos, cabe destacar que deriva de la clase *FileDataset* y a su vez de la clase *Dataset*, en este caso, el dataset es del formato Excel anteriormente detallado.
* Variables miembro:
* *bReadNamesAtStart:* si es verdadero, los nombres de las columnas serán leídos antes que las columnas.
* *iSkipStartLines:* variable que almacena el número de líneas del *dataset* al inicio que se desecharan en la lectura.
* *sInputColumnFormat:* variable usada para especificar los tipos de columna que contiene el *dataset*.
* *iNumFormatColumns:* número de columnas especificadas en el formato de columnas de la variable anterior.
* *bNeedReadNamesAtEnd*: si es verdadero, los nombres serán leidos al final del fichero.
* *iSkipEndLines*: número de líneas que se desechan de la lectura al final del fichero.
* *xExcelWorkBook*: almacena el archivo excel actualmente usado.
* *xExcelSheet*: almacena la hoja excel actualmente usada.
* *iActualRow*: variable que almacena el índice de la columna que está siendo usada actualmente.
* oFileReader: representa el flujo de entrada el archivo excel.
* bSheetOverwrite: si es verdadero y la ruta de destino de escritura del dataset es la misma que del origen, entonces la hoja se sobreescribirá, si es falso, se añadirá una nueva al archivo excel.
* iActualSheet: variable que almacena el índice de la hoja actualmente usada en el archivo excel.
* Métodos de la clase:
* *ExcelDataSet()*: constructor sin parámetros de la clase.
* *ExcelDataSet(String):* constructor parametrizado de la clase, que recibe la ruta del archivo a cargar.
* *open():* método que se encarga de obtener la cabecera del *dataset.*
* *close():* método que cierra el archivo del *dataset*.
* *reset():* método que realiza un reseteo de todas las variables asociadas a la lectura del *dataset* desde un fichero, para poder leerlo de nuevo.
* *obtainMetadata(String):* método que se encarga de realizar la lectura de la cabecera del *dataset*.
* *readDataset(String, String):* método que se encarga del proceso completo de lectura del *dataset*.
* *writeDataset(String):* método que realiza la escritura del contenido del dataset a un archivo en la ruta especificada como parámetro.
* *addAllValues():* método que realiza la lectura de los datos del *dataset*, y los añade a las columnas correspondientes en cada caso.
* *merge(Dataset):* método que realiza la unión del *dataset* pasado como parámetro al actual.
* *setFormatHeaderSpecification():* método que dada una especificación de los atributos en la variable *sInputColumnFormat*, añade todas las columnas vacías según el tipo especificado en la variable.
* *getOperationsArrayFromInput(String):* método que devuelve un array, donde cada elemento es una palabra clave de lo que hay que leer en cada momento, sirve para controlar el orden de lectura de los elementos del *dataset* (nombres, instancias o saltos de línea).
* *readNames(int):* método que realiza la lectura de los nombres del *dataset.*
* *setSheetOverwrite(boolean):* si el parámetro es verdadero, entonces la hoja del archivo excel se sobrescribirá en caso de que el *dataset* origen y destino sean los mismos.
* g*etSheetOverwrite():* devuelve el estado de la variable que sobrescribe la hoja en el archivo excel usado.
* *getActualSheet():* devuelve la hoja excel actualmente usada.

### Macintosh HD:Users:psylock:Desktop:ExcelDatasetClase.jpg

Figura 10.26: Clase *ExcelDataSet*

### Componente Controlador de estrategias

Este componente será el encargado de la definición de los algoritmos que se aplican sobre dataset, su estructura se muestra en la Figura 10.27.

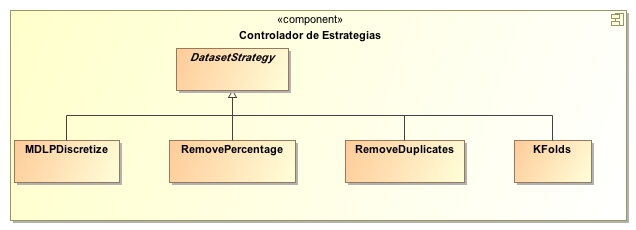


Figura 10.27: Componente Controlador de Estrategias

#### Clase DatasetStrategy

* Nombre: *DatasetStrategy*
* Descripción: es la clase encargada de definir la base para todos los algoritmos que se aplican sobre *datasets,* es decir, todos los algoritmos deberán derivar de esta clase base y usar su estructura.
* Variables miembro:
* *oDataset*: se trata del dataset al que se le aplicará el algoritmo.
* *bExecutable*: si es verdadero, el algoritmo es ejecutable, en caso contrario, no será ejecutable.
* Métodos de la clase:
* *isExecutable():* devuelve verdadero si el algoritmo es ejecutable, falso en caso contrario.
* *setExecutable(boolean):* modifica el valor de la variable bExecutable, al valor pasado como parámetro.
* *getDataset():* devuelve el *dataset* que se está usando como entrada del algoritmo.
* *setDataset(Dataset):* asigna el *dataset* que se usa como entrada.
* *initialize():* inicializa las variables necesarias para poder ejecutar el algoritmo.
* *execute():* ejecuta el algoritmo en sí.
* *postexec():* realiza una postejecución antes de realizar la obtención del resultado que ha generado el algoritmo.
* *getResult():* devuelve el resultado de la ejecución del algoritmo.

### Macintosh HD:Users:psylock:Desktop:PFC:Diagramas de clase:Controlador Estrategias:DatasetStategy.jpg

Figura 10.28: Clase *DatasetStrategy*

#### Clase MDLPDiscretize

* Nombre: *MDLPDiscretize*
* Descripción: es la clase encargada de definir el algoritmo de discretización MDLP, usa como clase base *DatasetStrategy*.
* Variables miembro:
* *aLinkedColumns*: se trata de una lista en la que se almacena otra lista, que contiene las columnas, almacenadas de tal forma, que cada uno de los valores es una tupla, cuyo primer elemento es el valor concreto de la columna, y el segundo es la categoría asociada a ese valor.
* *oDiscretization*: resultado de la discretización.
* *oSourceDataset*: variable que almacena el *dataset* origen.
* *oResultDataset*: variable que almacena el *dataset* resultado.
* Métodos de la clase:
* *MDLPDiscretize(Dataset):* constructor parametrizado de la clase, que recibe como parámetro el *dataset* que se usará como entrada del algoritmo.
* *initialize():* inicializa las variables necesarias para poder ejecutar el algoritmo.
* *execute():* ejecuta el algoritmo en sí.
* *getResult():* devuelve el resultado de la ejecución del algoritmo.
* *getCutPoints(List):* método que obtiene los puntos de corte de cada una de las columnas.
* *getMinEntropyCutPoint(List):* método que obtiene el punto de corte de mínima entropía para una partición dada.
* *logBase2(double):* devuelve el logaritmo en base dos de un número dado.
* *discretize():* método que realiza la discretización una vez obtenidos los punto de corte para cada una de las columnas.

### Macintosh HD:Users:psylock:Desktop:PFC:Diagramas de clase:Controlador Estrategias:MDLPDiscretize.jpg

Figura 10.29: Clase *MDLPDiscretize*

#### Clase KFolds

* Nombre: *KFolds*
* Descripción: es la clase encargada de definir el algoritmo de partición según KFolds.
* Variables miembro:
* *iNumberOfPartitions*: variable que almacena el número de particiones que se realizarán.
* *oSourceDataset*: variable que almacena el *dataset* origen.
* *oClassInstances*: tabla que almacena las instancias, de forma que se accede a cada una de ellas mediante su categoría.
* oDatasetPartitions: lista de *dataset*sdonde se almacenarán las particiones que se realicen del *dataset.*
* Métodos de la clase:
* *KFolds(Dataset):* constructor parametrizado de la clase, que recibe como parámetro el *dataset* que se usará como entrada del algoritmo.
* *initialize():* inicializa las variables necesarias para poder ejecutar el algoritmo.
* *execute():* ejecuta el algoritmo en sí.
* *getResult():* devuelve el resultado de la ejecución del algoritmo.

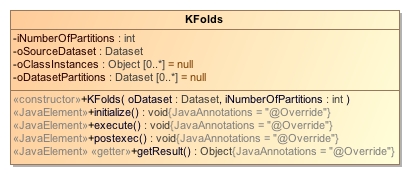


Figura 10.30: Clase *KFolds*

#### Clase RemoveDuplicates

* Nombre: *RemoveDuplicates*
* Descripción: es la clase encargada de definir el algoritmo mediante el cual se eliminan instancias duplicadas.
* Variables miembro:
* *oResultDataset*: variable que almacena el *dataset* resultado.
* *iActualInstanceIndex*: variable que almacena el índice de la instancia actual que se está usando.
* *iIndexesToRemove*: lista que almacena los índices de instancia que se deberán eliminar del *dataset.*
* Métodos de la clase:
* *RemoveDuplicates(Dataset):* constructor parametrizado de la clase, que recibe como parámetro el *dataset* que se usará como entrada del algoritmo.
* *initialize():* inicializa las variables necesarias para poder ejecutar el algoritmo.
* *execute():* ejecuta el algoritmo en sí.
* *getResult():* devuelve el resultado de la ejecución del algoritmo.
* *generateRemovalIndexes():* método que analiza el *dataset* en busca de instancias duplicadas y genera una lista con los índices de instancia que se deberán eliminar.
* *removeInstances():* método que a partir de la lista de índices a eliminar borra las instancias especificadas en dicha lista.

### Macintosh HD:Users:psylock:Desktop:PFC:Diagramas de clase:Controlador Estrategias:RemoveDuplicates.jpg

Figura 10.31: Clase *RemoveDuplicates*

#### Clase RemovePercentage

* Nombre: *RemovePercentage*
* Descripción: es la clase encargada de definir el algoritmo mediante el cual se elimina un porcentaje de instancias del *dataset*.
* Variables miembro:
* *oRnd*: variable que almacena el generador de números aleatorios.
* *iMode*: variable que almacena el modo de eliminación de instancias del *dataset*.
* *dPercentage*: variable que almacena el porcentaje de instancias a eliminar del *dataset.*
* iInstancesToRemove: variable que almacena el número de instancias que se borrarán.
* iIndexesToRemove: variable que almacena la lista de índices de las instancias a eliminar.
* Métodos de la clase:
* *RemovePercentage(Dataset, int, double):* constructor parametrizado de la clase, que recibe como parámetro el *dataset* que se usará como entrada del algoritmo, el modo de borrado de instancias y el porcentaje que se eliminará.
* *initialize():* inicializa las variables necesarias para poder ejecutar el algoritmo.
* *execute():* ejecuta el algoritmo en sí.
* *getResult():* devuelve el resultado de la ejecución del algoritmo.
* *nextRandomValidInteger():* devuelve un número aleatorio válido según el *dataset* de entrada.
* *generateRemovalIndexes():* método a partir del cual se obtienen los índices de las instancias que se eliminarán, según el modo y el porcentaje.
* *removeInstances():* método que a partir de los índices previamente generados, elimina las instancias del *dataset*.

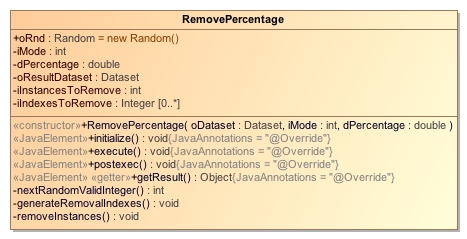


Figura 10.32: Clase *RemovePercentage*

### Componente Excepciones

La estructura de este componente se indica en la figura a continuación.



Figura 10.33: Componente Excepciones

#### Clase IllegalFormatSpecificationException

* Nombre: *IllegalFormatSpecificationException*
* Descripción: es la clase encargada de definir la excepción por la cual, el formato que se ha especificado de la forma de leer un *dataset* o de los tipos de columna que contiene un *dataset* contiene un símbolo no válido.
* Métodos de la clase:
* *IllegalFormatSpecificationException(String):* constructor parametrizado de la clase, que recibe como parámetro el mensaje de error que se quiere mostrar.

#### Clase NoSuchCategoryException

* Nombre: *NoSuchCategoryException*
* Descripción: es la clase encargada de definir la excepción por la cual, se lanza cuando se intenta usar/acceder a una categoría no existente en una columna de tipo categórico.
* Métodos de la clase:
* *NoSuchCategoryException(String):* constructor parametrizado de la clase, que recibe como parámetro el mensaje de error que se quiere mostrar.

#### Clase OutOfRangeException

* Nombre: *OutOfRangeException*
* Descripción: es la clase encargada de definir la excepción por la cual, se lanza cuando se intenta añadir un valor a una columna el cual está fuera de los límites permisibles de la misma.
* Métodos de la clase:
* *OutOfRangeException(String):* constructor parametrizado de la clase, que recibe como parámetro el mensaje de error que se quiere mostrar.

#### Clase NotAddedValue

* Nombre: *NotAddedValue.*
* Descripción: es la clase encargada de definir la excepción por la cual, se lanza cuando al añadir un valor a una columna, este por razones diversas no se ha podido añadir.
* Métodos de la clase:
* *NotAddedValue(String):* constructor parametrizado de la clase, que recibe como parámetro el mensaje de error que se quiere mostrar.

## Capa Interfaz de usuario

Esta capa es la que define la interfaz de usuario para las pruebas de los algoritmos desarrollados, incorpora cada una de las funcionalidades descritas en el capítulo 7.2.15.

### Componente Gestor E/S *Dataset*

Se especifica la estructura en la siguiente figura.

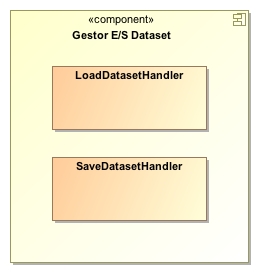


Figura 10.34: Componente Gestor E/S *Dataset*

#### Clase *LoadDatasetHandler*

* Nombre: *LoadDatasetHandler*
* Descripción: se trata de la clase que maneja la parte de la interfaz de usuario correspondiente a la carga de *datasets*.
* Métodos de la clase:
* LoadDatasetHandler(): constructor sin parámetros de la clase, no realiza ninguna acción.
* execute(ExecutionEvent): método que realiza la ejecución de la acción que recibe como parámetro, en este caso es dibujar la interfaz correspondiente a la carga del *dataset* y de definir las acciones que se producirán al interactuar con sus elementos.



Figura 10.35: Clase *LoadDatasetHandler*

#### Clase *SaveDatasetHandler*

* Nombre: *SaveDatasetHandler*
* Descripción: esta clase realiza la gestión necesaria para el guardado de *datasets.*
* Métodos de la clase:
* *SaveDatasetHandler():* constructor sin parámetros de la clase, no realiza ninguna acción.
* *execute(ExecutionEvent):* método que realiza la ejecución de la acción que recibe como parámetro, en este caso es dibujar la interfaz correspondiente al guardado del *dataset* y de definir las acciones que se producirán al interactuar con sus elementos.



Figura 10.36: Clase *SaveDatasetHandler*

### Componente Gestor Aplicación Base

Este componente es el encargado de definir la base para la aplicación de pruebas, proporciona métodos para añadir cada uno de los elementos que compondrán la interfaz final. No se realizará la descripción de su estructura y componentes, ya que es la propia que proporciona el entorno de desarrollo Eclipse[10] para desarrollar aplicaciones sobre la plataforma.

### Componente Gestor Operaciones y Estrategias

El Gestor de Operaciones y Estrategias, será el encargado de ofrecer la interfaz necesaria para el manejo de las operaciones aplicadas sobre columnas y las estrategias definidas por la librería sobre *datasets.* Su estructura es la siguiente.

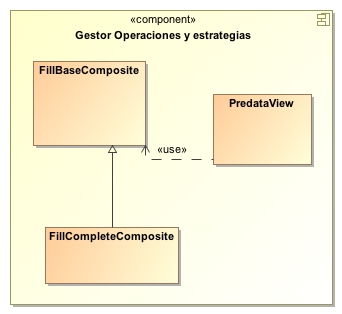


Figura 10.37: Componente Gestor Operaciones y estrategias

#### Clase *FillBaseComposite*

* Nombre: *FillBaseComposite*
* Descripción: es la clase encargada de definir la interfaz base donde se introducirán los controles adecuados para el manejo de las operaciones sobre columnas, además sirve como punto de extensión para poder ampliar la interfaz relacionada con las estrategias.
* Variables miembro:
* inputDatasetPath: almacena la ruta donde se encuentra el *dataset* de entrada.
* inputDataset: representa el *dataset* de entrada.
* outputDatasetPath: esta variable representa la ruta donde se guardará el *dataset* de salida.
* outputDataset: representa el *dataset* de salida.
* strategyTabFolder: esta variable representa el punto de extensión que permite modificar la interfaz de las estrategias.
* Métodos de la clase:
* FillBaseComposite(Composite, int): constructor parametrizado de la clase, que se encarga de la creación de la interfaz base del componente.
* refreshTable(): método que actualiza la tabla donde se mostrará el contenido del *dataset*.
* checkSubclass(): método que inhabilita el chequeo que hace SWT para que los widgets no puedan actuar como subclase.
* getInputDataset(): método que devuelve el *dataset* de entrada.
* setOutPutDataset(Dataset): método que establece como *dataset* de salida el pasado como parámetro.
* getStrategyTabFolder(): método que devuelve el punto de extensión de las estrategias.

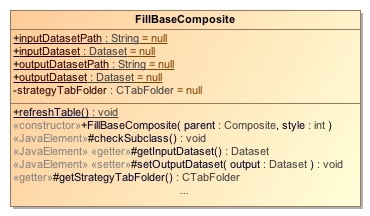


Figura 10.38: Clase *FillBaseComposite*

#### Clase *FillCompleteComposite*

* Nombre: *FillCompleteComposite*
* Descripción: clase encargada de definir la interfaz de las estrategias haciendo uso del punto de extensión de la superclase.
* Métodos de la clase:
* FillCompleteComposite(): constructor sin parámetros de la clase, que se encarga de definir todo el contenido de la interfaz de la estrategias, así como de definir su comportamiento al interactuar con sus elementos.



Figura 10.39: Clase *FillCompleteComposite*

### Componente Herramientas de recursos

Este componente será el encargado de aportar métodos útiles a la hora del manejo de archivos XML para la definición de las estrategias, y del manejo de archivos JAR que la interfaz de pruebas use en tiempo de ejecución. Su estructura se especifica en la siguiente figura.



Figura 10.40: Componente Herramientas de recursos

#### Clase *ClassLoaderResourcesUtilities*

* Nombre: *ClassLoaderResourcesUtilities*
* Descripción: esta clase aporta diferentes métodos para el manejo de archivos JAR y recursos que se encuentren en el entorno de ejecución.
* Métodos de la clase:
* getPackageClasses(String): método que devuelve la lista de clases existente en un paquete especificado como parámetro.
* getPackageXMLs(String): método que devuelve la lista de archivos XML que se encuentran en cualquier ubicación dentro del entorno de ejecución.
* getPackageFiles(String): devuelve una lista de archivos existentes en un paquete determinado, que es pasado como parámetro.
* getFilesFromJarFileByPackage(String, String, List): este método devuelve los archivos existentes en un paquete dentro de un archivo JAR.
* getDirectoryListOfFiles(String, List): realiza un escaneo en un directorio especificado como parámetro, y lo analiza recursivamente, obtienendo la lista completa de archivos existentes en el mismo.

#### Macintosh HD:Users:psylock:Desktop:PFC:Diagramas de clase:Interfaz:Herramientas de Recursos:ClassLoaderResourcesUtilities.jpg

Figura 10.41: Clase *ClassLoaderResourcesUtilities*

#### Clase XMLUtilities

* Nombre: *XMLUtilities*
* Descripción: esta clase aporta métodos útiles para el manejo de los archivos XML involucrados en la definición de las estrategias.
* Métodos de la clase:
* parseXMLStrategyDefinitions(ArrayList): este método a partir de la lista de archivos XML, analiza cada uno de ellos, para la extracción de la información de cada estrategia, nombre, descripción, parámetros, etc…

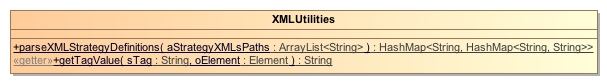


Figura 10.42: Clase *XMLUtilities*

Parte IV

PRUEBAS

# PRUEBA DE UNIDAD

La prueba de unidad será utilizada para comprobar que el sistema desarrollado cumple de manera correcta todas las especificaciones para las que fue definido. Estas especificaciones se comprobarán mediante dos técnicas diferentes, una externa, que verificará que toda la funcionalidad de la aplicación proporciona la salida esperada, y otra que verificará que existen distintos conjuntos de datos de entrada que hacen que se ejecutan todos los caminos posibles. La primera técnica se denomina Prueba de casos de uso, y en ella se realizarán pruebas para cada uno de los casos de uso descritos en el capítulo de análisis, y la segunda recibe el nombre de Pruebas de escenarios.

En el caso que ocupa este Proyecto, se ha diseñado una librería para el preprocesado de *datasets* y una interfaz de usuario para la prueba de las distintas operaciones implementadas en la librería, por tanto la interacción con los actores se realiza de dos formas diferentes, como queda reflejado en los casos de uso analizados en la Sección 7.2.1 de este documento. En primer lugar el desarrollador interactúa con la librería por medio de sus funciones de interfaz, y por tanto, la mayoría de los casos de prueba diseñados buscarán errores en la invocación y funcionamiento de esta funciones que proporciona la librería para su uso y su respuesta a posibles errores. Por otro lado, el usuario que haga uso de la librería por medio de la interfaz gráfica de usuario, a la hora hacer uso de las operaciones, se encontrará con un escenario completamente distinto y habrá que diseñar pruebas especialmente adaptadas a cada circunstancia.

## Pruebas de los casos de uso

Para llevar a cabo este tipo de pruebas, es necesario desarrollar una serie de casos de prueba que especificarán cómo probar cada caso de uso. Un caso de prueba de este tipo incluye la verificación del resultado de la interacción entre los actores y la aplicación, que se satisfagan las condiciones especificadas por dichos casos de uso y que se siga la secuencia de acciones especificada por el mismo.

Se puede observar que un caso de prueba basado en un caso de uso consiste claramente en una prueba del sistema como caja negra, es decir, una prueba del comportamiento del mismo observable desde el exterior. A continuación, se describirán los distintos casos de prueba que han sido diseñados para cada uno de los casos de uso de la aplicación.

### Caso de prueba Crear *Dataset*

* Entrada: invocación a la función constructora del tipo de *dataset* que se desea crear. Si se desea, el parámetro que se le podrá proporcionar al constructor será el siguiente.
* Ruta del *dataset* a cargar que se cargará posteriormente usando el método de carga.
* Salida: la función deberá proporcionar una referencia al *dataset* que permita trabajar con el objeto generado.
* Condiciones: el tipo de *dataset* que desea utilizarse deberá haber sido importado anteriormente.

### Caso de prueba Cargar *Dataset*

* Entrada: se invocará al método encargado de la carga del *dataset.* Especificando, si el formato lo requiere la forma de leer el *dataset* y el tipo de columnas que contiene.
* Salida: en caso de ocurrir algún error durante la lectura del *dataset,* se lanzará la excepción correspondiente.
* Condiciones: la creación del objeto *dataset* ha de ser anterior a la invocación de la función.

### Caso de prueba Crear Columna

* Entrada: se llamará a la función constructora del tipo de columna que se desea crear.
* Salida: el método deberá proporcionar una referencia a la columna que permita trabajar con el objeto generado.
* Condiciones: el tipo de columna que desea utilizarse deberá haber sido importado anteriormente.

### Caso de prueba Copiar *Dataset*

* Entrada: el desarrollador invocará a la función de copia del *dataset*
* Salida: la función debe proporcionar la referencia a un *dataset* que contiene la copia del original.
* Condiciones: el objeto *dataset* ha se de ser creado previamente.

### Caso de prueba Clonar *Dataset*

* Entrada: invocación a la función de clonación del *dataset*
* Salida: el método debe proporcionar la referencia a un *dataset* que contiene la estructura del original.
* Condiciones: el objeto *dataset* ha se de ser creado previamente.

### Caso de prueba Modificar columna

* Entrada: se invoca a alguna de las funciones de modificación de la columna, de metadatos o de contenido.
* Salida: el elemento es modificado según se haya indicado.
* Condiciones: se debe haber creado el objeto columna previamente.

### Caso de prueba Discretizar columna

* Entrada: se llamará a alguna de las funciones de discretización, especificando el parámetro que requiera según el método que se desee seguir.
* Salida: una columna de tipo rango, que contiene la discretización de la columna.
* Condiciones: el objeto columna ha se de ser creado previamente.

### Caso de prueba Añadir columna a *Dataset*

* Entrada: se invoca a la función que añade una columna al *dataset*, especificando como parámetro de la función, la columna que se desea añadir.
* Salida: el *dataset* se modifica, y ahora contiene la columna añadida.
* Condiciones: el objeto *dataset* y columna ha se de ser creado previamente.

### Caso de prueba Eliminar columna de *Dataset*

* Entrada: se realizará la invocación a la función que elimina una columna al *dataset*, especificando como parámetro de la función, el índice de la columna que se desea eliminar.
* Salida: el *dataset* ya no contiene la columna situada en el índice especificado.
* Condiciones: se debe crear el objeto *dataset* previamente, además, el índice que se especifique, debe estar dentro del rango 0 y número de columnas que contiene el *dataset* menos uno.

### Caso de prueba Consultar columna *Dataset*

* Entrada: se llamará al método encargado de la consulta de una columna en un *dataset*, se especificará como parámetro el índice de la columna que se desea consultar.
* Salida: el método devolverá la referencia a la columna consultada.
* Condiciones: el objeto *dataset* ha se de ser creado previamente, además, el índice especificado, debe estar dentro del rango 0 y número de columnas que contiene el *dataset* menos uno.

### Caso de prueba Intercambiar Columnas *Dataset*

* Entrada: el desarrollador invocará a la función de intercambio de columnas, especificando como parámetro de la función las referencias de las columnas que se desean intercambiar.
* Salida: el *dataset* se modifica de forma que las posiciones de ambas columnas son intercambiadas, en caso de que no exista alguna de las dos columnas en el *dataset*, no se realizará ninguna acción.
* Condiciones: el objeto *dataset* ha se de ser creado previamente y éste debe contener al menos dos columnas.

### Caso de prueba Guardar *dataset*

* Entrada: se lleva a cabo la llamada a la función de guardado del *dataset*, especificando como parámetro la ruta de guardado del mismo.
* Salida: el *dataset* se guarda en la ruta especificada, en caso de que ocurriese algún error, el *dataset* no se guarda, y se lanzará la excepción correspondiente al error acaecido.
* Condiciones: el objeto *dataset* ha se de ser creado previamente.

### Caso de prueba Ejecutar estrategia

* Entrada: se invoca a la función de ejecución de la estrategia.
* Salida: el objeto se modifica, albergando el resultado de la aplicación de la estrategia.
* Condiciones: el desarrollador ha de crear el objeto estrategia previamente e inicializarlo.

### Caso de prueba Crear proyecto

* Entrada: el usuario hace clic derecho sobre el explorador de proyectos y selecciona la opción de crear nuevo proyecto, especificando el nombre del mismo.
* Salida: un nuevo proyecto es creado.
* Condiciones: el nombre del proyecto debe ser diferente a los ya existentes.

### Caso de prueba Crear *Dataset* Nuevo

* Entrada: se hace clic derecho sobre un proyecto, seleccionando la opción de nuevo archivo, especificando un nombre para el mismo y la extensión.
* Salida: un nuevo *dataset* es creado.
* Condiciones: el nombre del *dataset* debe ser diferente a los ya existentes, además, debe existir al menos un proyecto en el espacio de trabajo.

### Caso de prueba Importar *Dataset*

* Entrada: el usuario hace clic derecho sobre un proyecto y selecciona la opción de importación, especificando en el sistema de archivos, qué archivo(s) importar al proyecto.
* Salida: los archivos seleccionados son añadidos al proyecto.
* Condiciones: los nombres de los archivos deben ser diferente a los ya existentes, además, debe existir al menos un proyecto en el espacio de trabajo.

### Caso de prueba Exportar *Dataset*

* Entrada: el usuario hará clic derecho sobre un archivo en el proyecto y selecciona la opción de exportar, especificando la ruta donde se guardarán los archivos exportados.
* Salida: los archivos seleccionados son exportados a la ubicación seleccionada.
* Condiciones: ninguna.

### Caso de prueba Cargar *Dataset* en Memoria

* Entrada: el usuario sobre el explorador de proyectos hará doble clic sobre un *dataset,* después de esto, selecciona desde la barra de menús la opción de carga, especificando los símbolos correspondientes a valores faltantes, nulos y vacíos, y si el formato lo requiere, la especificación acerca de la forma de leer el *dataset* además de la especificación del tipo de columnas que este contiene.
* Salida: el *dataset* es cargado en el espacio de trabajo, si hubiese algún error se indica mediante un cuadro de diálogo.
* Condiciones: debe existir al menos un proyecto y dentro de él, un *dataset* previamente creado o importado.

### Caso de prueba Aplicar operación sobre columna

* Entrada: el usuario selecciona una operación a aplicar sobre columnas, especificando sus parámetros de entrada, así como el rango de columnas a las que se le aplica la operación.
* Salida: las columnas son modificadas, en caso de ocurrir algún error, no se aplica la operación.
* Condiciones: el *dataset* debe haberse cargado previamente.

### Caso de prueba Escribir *Dataset*

* Entrada: se selecciona la opción de guardado desde la barra de menús, especifica el nombre de guardado y el formato del *dataset* destino.
* Salida: el *dataset* es guardado en el espacio de trabajo, si hubiese algún error se indica mediante un cuadro de diálogo.
* Condiciones: debe habersecargado previamente un *dataset*.

### Aplicar estrategia sobre *Dataset*

* Entrada: selección de una estrategia para aplicar sobre *dataset*, especificando los parámetros de entrada si los tuviese.
* Salida: el *dataset* es modificado, si hubiese algún error, no se aplica el algoritmo.
* Condiciones: el *dataset* debe haberse cargado previamente.

## Pruebas de la librería

Para la realización de este tipo de pruebas, es necesario llevar a cabo el diseño de una serie de caso de prueba. Un caso de prueba de este tipo puede incluir la verificación de la interacción entre los componentes que implementan un caso de uso. Los caos de prueba basados en una realización de caso de uso, generalmente especifican una prueba del sistema de caja blanca.

A continuación se describirán los distintos casos de prueba que han sido realizados para cada uno de los casos de uso de la librería.

### Caso de prueba Crear *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 1 | El usuario invoca al constructor del *dataset* sin ningún parámetro. | El *dataset* se crea con normalidad |
| 2 | El usuario invoca al constructor del *dataset* indicando una ruta incorrecta. | El dataset se crea con normalidad |

Tabla 11.1: Caso de prueba Crear *Dataset*

### Caso de prueba Cargar *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 3 | El usuario ha indicado una ruta incorrecta para la carga. | El sistema lanza una excepción informando de la situación. |
| 4 | El usuario ha indicado la ruta hacia un dataset de formato incorrecto. | El sistema lanza una excepción y el *dataset* no se carga. |

Tabla 11.2: Caso de prueba Cargar *Dataset*

### Caso de prueba Crear columna

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 5 | El usuario invoca al constructor de la columna sin ningún parámetro. | La columna se crea con normalidad. |
| 6 | El usuario invoca al constructor de la columna indicando su nombre. | La columna se crea con normalidad. |

Tabla 11.3: Caso de prueba Crear columna

### Caso de prueba Copiar *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 7 | Se invoca a la función de copia, cuando el *dataset* no contiene ninguna columna. | Se realiza la copia únicamente de los metadatos del *dataset.* |
| 8 | Se invoca a la función de copia, cuando el *dataset* contiene columnas, pero alguna de ellas no contiene ningún valor. | Se realiza la copia de lo que sea posible, aquellas columnas que no contengan valores, quedarán vacías. |
| 9 | Se invoca a la función de copia, el número de valores de alguna de las columnas es distintos al número de valores de todas las demás (suponiendo que sea igual) | Se realiza la copia normalmente, aquella columna que sea inconsistente, se copia tal cual al nuevo *dataset*. |

Tabla 11.4: Caso de prueba Copiar *Dataset*

### Caso de prueba Clonar *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 10 | El usuario llama al método de clonación y el *dataset* no contiene ninguna columna. | Se realiza la copia únicamente de los metadatos del *dataset.* |

Tabla 11.5: Caso de prueba Clonar *Dataset*

### Caso de prueba Modificar columna

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 11 | El usuario intenta añadir un valor a la columna, en una posición en la que ya existen datos. | El valor antiguo es reemplazado por el nuevo. |
| 12 | El usuario intenta añadir o modificar un valor de un tipo no compatible con el formato de columna. | El valor no es añadido, la función de modificación devuelve 0 y no se hace nada. |
| 13 | El usuario intenta añadir o modificar en un valor en un índice fuera de los límites de la columna. | El valor no es añadido, la función de modificación devuelve 0 y no se hace nada. |

Tabla 11.6: Caso de prueba Modificar columna

### Caso de prueba Discretizar columna

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 14 | El usuario llama al método de discretización especificando un parámetro incompatible con las características del discretizado, tipo y tamaño de columna | La discretización no se lleva a cabo y se lanza una excepción. |

Tabla 11.7: Caso de prueba Discretizar columna

### Caso de prueba Añadir columna a *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 15 | El usuario al llamar a la función de adición, le pasa como parámetro un valor nulo. | No se añade ninguna columna al *dataset.* |
| 16 | El usuario al llamar a la función, le pasa como parámetro la referencia de una columna ya existente en el *dataset*. | La columna es añadida y es independiente a la referenciada. |
| 17 | El usuario pasa como parámetro una columna que no contiene ningún valor. | La columna es añadida, pero esta no contendrá ningún valor. |

Tabla 11.8: Caso de prueba Añadir columna a *Dataset*

### Caso de prueba Eliminar columna de *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 18 | El usuario especifica un índice fuera de los límites de la lista de columnas del *dataset.* | El sistema lanza una excepción y no se realiza ningún cambio sobre el *dataset*. |

Tabla 11.9: Caso de prueba Eliminar columna de *Dataset*

### Caso de prueba Consultar columna de *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 19 | El usuario al llamar a la función de consulta se le especifica un índice fuera de los límites de la lista de columnas del *dataset*. | La función no devuelve nada y se lanza una excepción. |

Tabla 11.10: Caso de prueba Consultar columna de *Dataset*

### Caso de prueba Intercambiar columnas *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Descripción | | Acciones esperadas |
|  |  | |  |
| 20 | El usuario pasa como parámetro alguna columna que no existe en el *dataset.* | | No se realiza ninguna acción y el *dataset* queda inalterado. |
| 21 | | Se pasan dos referencias de columna iguales. | No se produce ningún cambio en el *dataset*. |

Tabla 11.11: Caso de prueba Intercambiar columnas *Dataset*

### Caso de prueba Guardar *dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 22 | El usuario especifica una ruta de guardado inexistente. | No se realiza el guardado del *dataset* y el sistema lanza una excepción. |
| 23 | El usuario no ha especificado ningún símbolo para valores nulos, faltantes o vacíos. | Se realiza el guardado del *dataset* usando los valores por defecto que tienen asignados dichos símbolos. |
| 24 | El usuario especifica en alguno de los símbolos usados para valores nulos, faltantes o vacíos, una cadena de texto vacía. | Si escribe el *dataset* usando la cadena de caracteres vacía como símbolo. |
| 25 | Una columna del *dataset* tiene un número de valores distinto al número de valores que tienen las demás columnas. | No se realiza el guardado del *dataset* y se lanza una excepción. |
| 26 | No existe ninguna columna en el *dataset* a la hora del guardado. | No se realiza el guardado del *dataset* y se lanza una excepción. |
| 27 | Existen columnas, pero todas están vacías. | Se realiza el guardado del *dataset* pero solamente se escribirá su cabecera, no conteniendo este ningún valor. |

Tabla 11.12: Caso de prueba Guardar *dataset*

### Caso de prueba Ejecutar estrategia

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 28 | El usuario llama a la función de ejecución sin haber antes inicializado la estrategia. | No se realiza ninguna acción y el resultado no esta disponible. |
| 29 | El usuario al crear la estrategia ha definido parámetros incompatibles con el *dataset*. | No se realiza la ejecución del algoritmo y el resultado no está disponible. |

Tabla 11.13: Caso de prueba Ejecutar estrategia

### Caso de prueba Crear proyecto

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 30 | El usuario especifica como nombre de proyecto el de uno ya existente. | El sistema detecta esta situación e impide la creación del nuevo proyecto. |

Tabla 11.14: Caso de prueba Crear proyecto

### Caso de prueba Crear *Dataset* Nuevo

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 31 | El usuario especifica como nombre de *dataset* el de uno ya existente en el proyecto seleccionado. | El sistema detecta esta situación e impide la creación del *dataset*. |

Tabla 11.15: Caso de prueba Crear *Dataset* Nuevo

### Caso de prueba Importar *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 32 | El usuario importa un *dataset* que no contiene nada. | El archivo es importado sin problemas. |
| 33 | El usuario importa un *dataset* sin extensión. | El archivo es importado sin problemas. |

Tabla 11.16: Caso de prueba Importar *Dataset*

### Caso de prueba Exportar *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 34 | El usuario selecciona para exportar un *dataset* vacío. | El archivo se exporta sin problemas. |
| 35 | El usuario exporta un *dataset* sin extensión. | El archivo se exporta sin problemas. |

Tabla 11.17: Caso de prueba Exportar *Dataset*

### Caso de prueba Cargar *Dataset* en Memoria

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Descripción | | Acciones esperadas |
|  |  | |  |
| 36 | El usuario no ha seleccionado ningún *dataset* para cargar. | | El sistema informa de la situación mediante cuadro de diálogo. |
| 37 | | El usuario selecciona como formato uno que no coincide con el que realmente tiene el *dataset* seleccionado. | El sistema informa de la situación mediante cuadro de diálogo. |
| 38 | | El usuario indica símbolos de valores vacíos, faltantes o nulos, incorrectos para el *dataset* que se está cargando. | El sistema informará en caso de conflicto de la situación mediante cuadro de diálogo. |
| 39 | | El usuario realiza una especificación del tipo de columnas a cargar incorrecto. | El sistema informa de la situación mediante cuadro de diálogo. |
| 40 | | El usuario realiza una especificación de la forma de leer el *dataset* incorrecta. | El sistema informa de la situación mediante cuadro de diálogo. |

Tabla 11.18: Caso de prueba Cargar *Dataset*

### Caso de prueba Aplicar operación sobre columna

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 41 | El usuario especifica parámetros incorrectos para la operación a aplicar. | El sistema informa de la situación mediante cuadro de diálogo. |
| 42 | El usuario especifica un rango de índices a los que se aplicará la operación incorrecto. | El sistema informa de la situación mediante cuadro de diálogo. |
| 43 | Se intenta aplicar una operación sobre un tipo de columna incompatible. | El sistema no aplica la operación sobre dicha columna, e informa del número de columnas a las que se le aplicó correctamente la operación. |

Tabla 11.19: Caso de prueba Aplicar operación sobre columna

### Caso de prueba Escribir *Dataset*

|  |  |  |
| --- | --- | --- |
| Caso | Descripción | Acciones esperadas |
|  |  |  |
| 44 | El usuario especifica como nombre de guardado el mismo de un *dataset* ya existente. | El *dataset* se guarda sobrescribiendo el antiguo. |
| 45 | El usuario selecciona como formato de guardado uno distinto al formato de origen con el que se cargó el *dataset*. | Se realiza un cambio de formato y se guarda con el nombre especificado. |

Tabla 11.20: Caso de prueba Guardar *Dataset*

### Caso de prueba Aplicar estrategia sobre *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Descripción | | Acciones esperadas |
|  |  | |  |
| 46 | El usuario especifica un parámetro incompatible. | | El sistema informa de la situación mediante cuadro de diálogo y no se aplica el algoritmo. |
| 47 | | El usuario intenta aplicar el algoritmo a un tipo de *dataset* incompatible. | El sistema informa de la situación mediante cuadro de diálogo y no se aplica el algoritmo. |

Tabla 11.21: Caso de prueba Aplicar estrategia sobre *Dataset*

# PRUEBA DE LA APLICACIÓN

En el presente capítulo se procederá a describir los resultados obtenidos de la realización de los casos y procedimientos de prueba diseñados en el capítulo anterior, con el objetivo de probar los casos de uso más importantes. Para la ejecución de estos casos de prueba, la librería ha sido puesta en explotación durante un periodo de tiempo, en un ambiente real de trabajo.

## Proceso de pruebas

Para la prueba de los distintos casos, se ha puesto en explotación un caso real del sistema, generando el código necesario para cumplir fielmente cada uno de los casos de aplicación diseñados en el capítulo anterior. Para cada uno de ellos a sido necesario generar un código de inicialización de la aplicación que cumpla con las precondiciones que establecían y ejecutar la aplicación para el código generado, observando la respuesta de la librería de cara al usuario y más tarde con el depurador, de observarse alguna anomalía, para así poder identificar el origen del problema.

### Ejecución de los casos de prueba de la librería

En las tablas que aparecerán en los apartados siguientes, se podrá examinar el comportamiento observado en la aplicación durante las situaciones de posible error que se produjeron en el transcurso de la utilización de la misma. Las tablas están compuestas por tres columnas de datos que indican la situación de error que se produjo en el sistema software desarrollado, el resultado de la operación llevada a cabo por la aplicación como consecuencia de dicho error y el estado de la misma.

Se mostrará *Correcto* o *Fallido*, dependiendo si el resultado de la prueba ha sido satisfactorio o no, y en caso negativo será documentada su solución en secciones posteriores.

### Caso de prueba Crear *Dataset*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso | Situación de error | | Resultado | Estado |
|  |  | |  |  |
| 1 | El usuario invoca al constructor del *dataset* sin ningún parámetro. | | Completado | Correcto |
| 2 | | El usuario invoca al constructor del *dataset* indicando una ruta incorrecta. | Completado | Correcto |

Tabla 12.1: Caso de prueba Crear *Dataset*

### Caso de prueba Cargar *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 3 | El usuario ha indicado una ruta incorrecta para la carga. | Completado | Correcto |
| 4 | El usuario ha indicado la ruta hacia un dataset de formato incorrecto. | Completado | Correcto |

Tabla 12.2: Caso de prueba Cargar *Dataset*

### Caso de prueba Crear columna

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 5 | El usuario invoca al constructor de la columna sin ningún parámetro. | Completado | Correcto |
| 6 | El usuario invoca al constructor de la columna indicando su nombre. | Completado | Correcto |

Tabla 12.3: Caso de prueba Crear columna

### Caso de prueba Copiar *Dataset*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado | |
|  |  |  |  | |
| 7 | Se invoca a la función de copia, cuando el *dataset* no contiene ninguna columna. | Completado | Correcto | |
| 8 | Se invoca a la función de copia, cuando el *dataset* contiene columnas, pero alguna de ellas no contiene ningún valor. | Completado | Correcto | |
| 9 | Se invoca a la función de copia, el número de valores de alguna de las columnas es distintos al número de valores de todas las demás (suponiendo que sea igual) | Completado | | Correcto |

Tabla 12.4: Caso de prueba Copiar *Dataset*

### Caso de prueba Clonar *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  | |
| 10 | El usuario llama al método de clonación y el *dataset* no contiene ninguna columna. | Completado | Correcto |

Tabla 12.5: Caso de prueba Clonar *Dataset*

### Caso de prueba Modificar columna

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 11 | El usuario intenta añadir un valor a la columna, en una posición en la que ya existen datos. | Completado | Error |
| 12 | El usuario intenta añadir o modificar un valor de un tipo no compatible con el formato de columna. | Completado | Error |
| 13 | El usuario intenta añadir o modificar en un valor en un índice fuera de los límites de la columna. | Completado | Correcto |

Tabla 12.6 Caso de prueba Modificar columna

### Caso de prueba Discretizar columna

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 14 | El usuario llama al método de discretización especificando un parámetro incompatible con las características del discretizado, tipo y tamaño de columna | Completado | Correcto |

Tabla 12.7: Caso de prueba Discretizar columna

### Caso de prueba Añadir columna a *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 15 | El usuario al llamar a la función de adición, le pasa como parámetro un valor nulo. | Completado | Correcto |
| 16 | El usuario al llamar a la función, le pasa como parámetro la referencia de una columna ya existente en el *dataset*. | Completado | Error |
| 17 | El usuario pasa como parámetro una columna que no contiene ningún valor. | Completado | Correcto |

Tabla 12.8: Caso de prueba Añadir columna a *Dataset*

### Caso de prueba Eliminar columna de *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 18 | El usuario especifica un índice fuera de los límites de la lista de columnas del *dataset.* | Completado | Correcto |

Tabla 12.9: Caso de prueba Eliminar columna de *Dataset*

### Caso de prueba Consultar columna de *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 19 | El usuario al llamar a la función de consulta se le especifica un índice fuera de los límites de la lista de columnas del *dataset*. | Completado | Correcto |

Tabla 12.10: Caso de prueba Consultar columna de *Dataset*

### Caso de prueba Intercambiar columnas *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 20 | El usuario pasa como parámetro alguna columna que no existen el *dataset.* | Completado | Correcto |
| 21 | Se pasan dos referencias de columna iguales. | Completado | Correcto |

Tabla 12.11: Caso de prueba Intercambiar columnas *Dataset*

### Caso de prueba Guardar *dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 22 | El usuario especifica una ruta de guardado inexistente. | Completado | Correcto |
| 23 | El usuario no ha especificado ningún símbolo para valores nulos, faltantes o vacíos. | Completado | Correcto |
| 24 | El usuario especifica en alguno de los símbolos usados para valores nulos, faltantes o vacíos, una cadena de texto vacía. | Completado | Correcto |
| 25 | Una columna del *dataset* tiene un número de valores distinto al número de valores que tienen las demás columnas. | Completado | Correcto |
| 26 | No existe ninguna columna en el *dataset* a la hora del guardado. | Completado | Correcto |
| 27 | Existen columnas, pero todas están vacías. | Completado | Correcto |

Tabla 12.12: Caso de prueba Guardar *dataset*

### Caso de prueba Ejecutar estrategia

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 28 | El usuario llama a la función de ejecución sin haber antes inicializado la estrategia. | Completado | Correcto |
| 29 | El usuario al crear la estrategia ha definido parámetros incompatibles con el *dataset*. | Completado | Correcto |

Tabla 12.13: Caso de prueba Ejecutar estrategia

### Caso de prueba Crear proyecto

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 30 | El usuario especifica como nombre de proyecto el de uno ya existente. | Completado | Correcto |

Tabla 12.14: Caso de prueba Crear proyecto

### Caso de prueba Crear *Dataset* Nuevo

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 31 | El usuario especifica como nombre de *dataset* el de uno ya existente en el proyecto seleccionado. | Completado | Correcto |

Tabla 12.15: Caso de prueba Crear *Dataset* Nuevo

### Caso de prueba Importar *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 32 | El usuario importa un *dataset* que no contiene nada. | Completado | Correcto |
| 33 | El usuario importa un *dataset* sin extensión. | Completado | Correcto |

Tabla 12.16: Caso de prueba Importar *Dataset*

### Caso de prueba Exportar *Dataset*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso | Situación de error | | Resultado | Estado |
|  |  | |  |  |
| 34 | El usuario selecciona para exportar un *dataset* vacío. | | Completado | Correcto |
| 35 | | El usuario exporta un *dataset* sin extensión. | Completado | Correcto |

Tabla 12.17: Caso de prueba Exportar *Dataset*

### Caso de prueba Cargar *Dataset* en Memoria

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 36 | El usuario no ha seleccionado ningún *dataset* para cargar. | Completado | Error |
| 37 | El usuario selecciona como formato uno que no coincide con el que realmente tiene el *dataset* seleccionado. | Completado | Correcto |
| 38 | El usuario indica símbolos de valores vacíos, faltantes o nulos, incorrectos para el *dataset* que se está cargando. | Completado | Correcto |
| 39 | El usuario realiza una especificación del tipo de columnas a cargar incorrecto. | Completado | Correcto |
| 40 | El usuario realiza una especificación de la forma de leer el *dataset* incorrecta. | Completado | Correcto |

Tabla 12.18: Caso de prueba Cargar *Dataset*

### Caso de prueba Aplicar operación sobre columna

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 41 | El usuario especifica parámetros incorrectos para la operación a aplicar. | Completado | Correcto |
| 42 | El usuario especifica un rango de índices de columnas a las que se aplicará la operación incorrecto. | Completado | Error |
| 43 | Se intenta aplicar una operación sobre un tipo de columna incompatible. | Completado | Correcto |

Tabla 12.19: Caso de prueba Aplicar operación sobre columna

### Caso de prueba Escribir *Dataset*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso | Situación de error | | Resultado | Estado |
|  |  | |  |  |
| 44 | El usuario especifica como nombre de guardado el mismo de un *dataset* ya existente. | | Completado | Correcto |
| 45 | | El usuario selecciona como formato de guardado uno distinto al formato de origen con el que se cargó el *dataset*. | Completado | Correcto |

Tabla 12.20: Caso de prueba Guardar *Dataset*

### Caso de prueba Aplicar estrategia sobre *Dataset*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 46 | El usuario especifica un parámetro incompatible. | Completado | Correcto |
| 47 | El usuario intenta aplicar el algoritmo a un tipo de *dataset* incompatible. | Completado | Correcto |

Tabla 12.21: Caso de prueba Aplicar estrategia sobre *Dataset*

## Resultado de las pruebas

Una vez aplicadas las distintas pruebas que se han diseñado en los capítulos anteriores y habiendo observado los resultados obtenidos tras su aplicación, se ha concluido que no existía ninguna anomalía asociada a la ejecución de cada uno de los casos, en cada uno de ellos se ha obtenido el resultado que se esperaba.

Como se ha podido comprobar en el apartado anterior, la librería no actuaba de forma correcta en cinco casos, lo que provocaba cierta inestabilidad. Los fallos detectados fueron.

* Caso 11. No se puede modificar el valor de una posición en concreto dentro de una columna.
* Caso 12. Es posible añadir un valor de tipo incompatible a una columna.
* Caso 16. Al añadir una columna usando la referencia de una columna del *dataset* sigue existiendo dependencia entre las dos.
* Caso 36. Es posible cargar un *dataset* desde la interfaz gráfica de usuario sin haber seleccionado un *dataset*.
* Caso 42. Es posible seleccionar un rango incorrecto para la aplicación de una operación sobre columnas en la interfaz.

## Solución de los problemas detectados

Analizando los errores detectados, se comprueban que todos ellos son controlables. A continuación se muestran los cambios llevados a cabo según el problema detectado.

### Solución del Caso 11

En este caso, se esperaba que se pudiese añadir un valor a una posición concreta de una columna. Por el contrario, lo que ocurría es que al añadir el valor en la posición indicada, resultaba en un desplazamiento de todos los elementos que ocupasen un índice superior al indicado. Esta operación era realizada mediante el método *addValue(Object, int)*.

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 11 | El usuario intenta añadir un valor a la columna, en una posición en la que ya existen datos. | Completado | Correcto |

La solución en este caso, sería añadir a las clases *ColumnAbstraction* y *ColumnImpl* el método *setValue(Object, int)*, el cual recibirá el valor a añadir como parámetro y el índice donde situar dicho valor.

Tabla 12.22: Resultado de la corrección del caso 11

Este método ha debido añadirse a todas las columnas. La mayoría de las columnas, para almacenar sus datos, constan de un array, este array es de la clase *List*, por lo que en el cuerpo de este método se hará uso del método *set* que proporciona dicha clase, en lugar de *add.*

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 12 | El usuario intenta añadir o modificar un valor de un tipo no compatible con el formato de columna. | Completado | Correcto |

### Solución del Caso 12

Este caso se esperaba en un principio que no supusiera ningún problema el hecho de añadir un valor de tipo incompatible a las columnas, es por ello que es necesario el control en el cuerpo de cada función de adición o modificación de valores en las columnas.

Java ofrece la posibilidad de realizar un chequeo de la clase a la que pertenece un objeto mediante *instanceof*, por lo que al inicio de cada método de adición o modificación ha sido necesario realizar la comprobación de la clase del objeto recibido.

Tabla 12.23: Resultado de la corrección del caso 12

### Solución del Caso 16

En este caso, se esperaba que tras la adición de una nueva columna al *dataset,* siendo el parámetro pasado una referencia a una columna ya existente en el mismo, se añadiera una copia de la columna y no una referencia, por lo que existiría dependencia entre una y otra.

Es posible solucionar esto, haciendo que el método de adición de columnas, antes de añadir la nueva columna a las ya existentes, realice una copia de la recibida como parámetro, para posteriormente añadirla al *dataset*, con ello se ha subsanado el error.

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 16 | El usuario al llamar a la función, le pasa como parámetro la referencia de una columna ya existente en el *dataset*. | Completado | Correcto |

Tabla 12.24: Resultado de la corrección del caso 16

### Solución del caso 36

Este error es asociado a la interfaz de pruebas, se esperaba que no se pudiese realizar la carga de un *dataset* sin haberlo seleccionado con anterioridad, para solucionar esta situación, en el método que ejecuta la apertura de la ventana de carga, se ha añadido al principio la condición de que si no existe *dataset* seleccionado, se muestre un mensaje indicándolo impidiendo de esta forma la apertura de dicha ventana.

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 36 | El usuario no ha seleccionado ningún *dataset* para cargar. | Completado | Correcto |

Tabla 12.25: Resultado de la corrección del caso 36

### Solución del caso 42

A la hora de aplicar una operación sobre columnas, es posible seleccionar el rango de índices sobre los que se aplicará la operación, se esperaba que si se seleccionase un rango superior al número de columnas existentes en el *dataset* no se aplicase la operación. Esto es posible solucionarlo, mediante la comprobación antes de aplicar la operación, de los índices que el usuario ha introducido, de forma que, el límite superior no debe sobrepasar el número de columnas que el *dataset* contiene.

|  |  |  |  |
| --- | --- | --- | --- |
| Caso | Situación de error | Resultado | Estado |
|  |  |  |  |
| 42 | El usuario especifica un rango de índices de columnas a las que se aplicará la operación incorrecto. | Completado | Correcto |

Tabla 12.26: Resultado de la corrección del caso 42

Como conclusión, se puede demostrar que la librería implementada cumple de manera satisfactoria todos los requisitos para los cuales fue diseñada, obteniéndose así una librería estable ante múltiples situaciones de posible error.

Parte V

CONCLUSIONES Y FUTURAS MEJORAS

# CONCLUSIONES Y FUTURAS MEJORAS

Una vez concluidas todas las fases planificadas para el desarrollo del sistema software, se realizará una exposición de las conclusiones a las que ha llegado después de la finalización del presente Proyecto. Estas conclusiones analizan los resultados obtenidos respecto a los objetivos planteados al comienzo de este documento y los resultados de la fase de pruebas.

## Conclusiones sobre los objetivos planteados

Se puede afirmar que los objetivos planteados al inicio del proyecto se han cumplido, expondremos los más importantes logros a modo de resumen.

* Se ha logrado diseñar una arquitectura para el manejo de diferentes tipos de *dataset* así como de varios tipos de columnas, además de la incorporación de diferentes operaciones relacionadas con el preprocesado y algoritmos complejos sobre *datasets*.
* Se ha logrado un manejo simple de valores nulos, vacíos y faltantes.
* Se ha conseguido una librería altamente extensible, ya que su estructura de clases, permite la fácil ampliación de los distintos formatos de *datasets* implementados, y además de los diferentes tipos de columna.
* Se ha diseñado un método por el cual se pueden añadir algoritmos aplicables sobre *datasets*, que ofrece una estructura predefinida para de esta forma, homogeneizar el aspecto externo de cada uno de los algoritmos que pudieran implementarse.
* Se ha logrado la implementación de diferentes algoritmos de discretización básicos como *equal-width, equal-frequency, by-size, by-amplitude* además de un algoritmo de discretización basado en entropía.
* Se ha logrado la implementación de algoritmos de particionado como KFolds[9], un algoritmo de eliminación de instancias repetidas y un algoritmo de eliminación de porcentajes de instancias del *dataset.*
* Se ha logrado la implementación de los seis tipos de columnas propuestos en el Capítulo 3 de este documento.

## Conclusión personal del autor

Aún siendo la fase de pruebas satisfactoria, no cabe ninguna duda, de que alguna de las partes de la librería podría haber sido mejor implementada, no obstante, parece ser que la estructura que se ha seguido para su diseño ha sido la idónea desde el punto de vista de la extensibilidad de la misma, así como de su uso. Una de las características más novedosas y útiles de la librería es la carga de *datasets* en memoria albergando una estructura totalmente común a todos los formatos, lo que permite una ‘traducción’ directa a otros formatos de forma sencilla y rápida.

El no conocimiento al inicio del Proyecto de la materia y la evolución acerca del aprendizaje del tema conforme se ha ido desarrollando, ha resultado mucho mejor de lo que pudiera haber imaginado al comienzo del mismo.

## Futuras mejoras

Los objetivos expuestos al comienzo del Proyecto daban solución al sistema que se pretendía desarrollar de una forma muy completa y como demostración de ello ha sido el resultado obtenido, sin embargo, durante el desarrollo del mismo aparecían nuevas ideas que por no ser de importancia o salirse de la temática del proyecto no se contemplaron, siendo interesante reflejarlas ahora, como futuras mejoras, se detallan a continuación las más relevantes.

* Gestión de *datasets* que contengan gran número de instancias, este tipo de *dataset* necesitan de una estructura o mecanismo para su manejo ya que no es posible volcarlos a memoria por completo, por lo que podría ser necesario cargarlo por partes.
* Posibilidad de realizar el manejo de *dataset* situados en diferentes archivos como si de uno solo se tratase.
* Adición de más formatos de *datasets*, tipos de columnas, operaciones sobre columnas y estrategias aplicables sobre *datasets*.
* Eliminar la dependencia existente con Apache POI para el manejo de archivos provenientes de la plataforma Microsoft Office para la carga de *datasets* en formato Excel.

Todas estas mejoras han sido propuestas para añadirse en próximas versiones de la aplicación.

BIBLIOGRAFÍA

[1] Pyle, D., 1999. *Data Preparation for Data Mining.* Morgan Kaufmann Publishers, Los Altos, California.

[2] Lui H. Et al. *Discretization: An Enabling Technique*. Data Mining and Knowledge Discovery, Vol 6 2002. pp. 393-423.

[3] Kotsiantis S. Kanellopoulos D. *Discretization Techniques:* *A recent survey*. GESTS International Transactions on Internet Technology. Vol. 32. Octubre 2006. pp. 47-58.

[4] Usama M. Fayyad, Keki B. Irani. *Multi-interval discretization of continuous valued attributes for classification learning*. In: Thirteenth International Joint Conference on Artificial Intelligence, 1022-1027, 1993.

[5] Alcalá-Fernández, Jesús; María José del: Garrel, Josep M.; et al. Proyecto KEEL*: Desarrollo de una Herramienta para el Análisis e Implementación de Algoritmos de Extracción de Conocimiento Evolutivos*.

[6] Web Weka: <http://www.cs.waikato.ac.nz/ml/weka/> [Última visita: 05-08-2011]

[7] Comma Separated Values Specification: <http://tools.ietf.org/html/rfc4180> [Última visita 05-08-2011]

[8] Microsoft Office Binary File Formats: <http://www.microsoft.com/interop/docs/officebinaryformats.mspx> [Última visita 05-08-2011]

[9] Kohavi, Ron (1995). *“A study of cross-validation and boostrap for accuracy Estimation and model selection*”. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence 2 (12): 1137–1143.(Morgan Kaufmann, San Mateo)

[10] Eclipse Platform: <http://www.eclipse.org/platform/> [Última visita 06-08-2011]

[11] Java Virtual Machine <http://www.java.com/en/> [Última visita 06-08-2011]

[12] The Apache POI Project <http://poi.apache.org/> [Última visita 06-08-2011]

[13] Web Knime <http://www.knime.org/> [Última visita 06-08-2011]

[14] Web RapidMiner <http://rapid-i.com/> [Última visita 06-08-2011]

[15] Web NetBeans <http://netbeans.org/> [Última visita 06-08-2011]

[16] Hungarian Notation <http://msdn.microsoft.com/en-us/library/aa260976.aspx> [Última visita 06-08-2011]

[17] JavaDoc Tool Home Page <http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html> [Última visita 06-08-2011]

[18] Web SWT/Jface <http://www.eclipse.org/swt> [Última visita 07-08-2011]

[19] Schmidberger G. Frank E. *Unsupervised Discretization using Tree-based Density Estimation.* Deparment of Computer Science (University of Waikato) Hamilton, New Zealand.

[20] M.-A. Krogel and S. Wrobel. *Transformation-Based Learning Using Multirelational Aggregation.* In C. Rouveirol and M. Sebag, editors, Proceedings of the Eleventh International Conference on Inductive Logic Programming(ILP), LNAI 2157, pp. 142-155. Springer, 2001.

[21] J.S.Vitter, *Random Sampling with a Reservoir*. ACM Transactions on Mathematical Software, pp. 37-57, Brown University, 1985.

[22] Igor Kononenko, *On Biases in Estimating Multi-Valued attributes*. In: 14th International Joint Conference on Artificial Intelligence, 1034-1040, 1995.

[23] Nitesh V. Chawla et. al. (2002). *Synthetic Minority Over-sampling Technique.* Journal of Artificial Intelligence Research. 16:321-357.

[24] Dougherty, J.; Kohavi, R.; and Sahami, M. 1995. *Supervised and unsupervised discretization of continuous features*. In Machine Learning: Proceedings of the Twelfth Int. Conference, 194-202. Morgan Kauf- mann.

[25] Alfred O. Hero; Gilles Fleury. *Pareto-Optimal Methods for Gene Analysis.* Univeristy of Michigan, Ann Arbor, Mi and Ecole Supeérieure d’Electricité, Gif-sur-Yvette, France. May 21, 2003.

[26] Kurgan, L., & Cios, K.J., CAIM Discretization Algorithm, *IEEE Transactions of Knowledge and Data Engineering*, accepted, 2003.

[27] Web Java-ML <http://java-ml.sourceforge.net/> [Última visita 24-08-2011]

[28] Web JDMP <http://www.jdmp.org/> [Última visita 24-08-2011]

[29] Unified Modelling Language Web Page <http://www.uml.org/> [Última visita 08-08-2011]