

RapidMiner: Tutorial online + Operadores

Copyright © 2010 Leonardo M. Tito, Felipe Mullicundo. Se otorga permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, Versión 1.3 o cualquier otra versión posterior publicada por la Free Software Foundation. Una copia de la licencia se puede encontrar en el siguiente enlace: <http://www.gnu.org/licenses/fdl-1.3.html>

| FECHA | AUTOR/A | VERSION | DETALLE |
|------------|-------------------------------------|---------|-------------------|
| 22/08/2010 | Leonardo M. Tito, Felipe Mullicundo | 0.1 | Verión Inicial. |
| 12/10/2010 | Ing. Bernabeu Ricardo Dario | 0.2 | Revisión general. |

Para ver la traducción no oficial de la Licencia de Documentación Libre de GNU, seguir el siguiente enlace: http://stuff.danexnow.org/gfdl_es.html

INDICE

- [Instalación.](#)
- [Tutorial de RapidMiner 5.0](#)
 - [Ejemplo 1: Árbol de Decisión.](#)
 - [Ejemplo 2: Reglas de Asociación.](#)
 - [Ejemplo 3: Stacking.](#)
 - [Ejemplo 4: K-Medias.](#)
 - [Ejemplo 5: Visualización de SVM.](#)
 - [Ejemplo 6: Rellenado de valores faltantes.](#)
 - [Ejemplo 7: Generador de ruido.](#)
 - [Ejemplo 8: Unión de Conjuntos de Ejemplos.](#)
 - [Ejemplo 9: Validación Cruzada Numérica.](#)
 - [Ejemplo 10: Aprendizaje sensitivo al costo y gráfico ROC.](#)
 - [Ejemplo 11: Aprendizaje de Costos Asimétricos.](#)
 - [Ejemplo 12: Aprendizaje Sensible al Costo.](#)
 - [Ejemplo 13: Análisis de Componentes Principales.](#)
 - [Ejemplo 14: Selección Forward.](#)
 - [Ejemplo 15: Selección Multiobjetivos.](#)
 - [Ejemplo 16: Validación Wrapper.](#)
 - [Ejemplo 17: YAGGA.](#)
 - [Ejemplo 18: Configuración atributos resultantes de YAGGA.](#)
 - [Ejemplo 19: Generación de Características Definidas por el Usuario.](#)
 - [Ejemplo 20: Ponderación Evolutiva.](#)
 - [Ejemplo 21: Visualización del Conjunto de Datos y Pesos.](#)
 - [Ejemplo 22: Optimización de Parámetros.](#)
 - [Ejemplo 23: Habilitador de Operadores.](#)
 - [Ejemplo 24: Umbral de Ponderación.](#)
 - [Ejemplo 25: Prueba de Significancia.](#)
 - [Ejemplo 26: Cálculos Basados en Grupos.](#)
- [Anexo: Descripción de los Operadores utilizados en el Tutorial de RM5](#)
 - [1. Data Transformation → Aggregation → Aggregate](#)
 - [2. Data Transformation → Attribute Set Reduction and Transformation → Generation → Generate Attributes](#)
 - [3. Data Transformation → Attribute Set Reduction and Transformation → Generation → Generate ID](#)
 - [4. Data Transformation → Attribute Set Reduction and Transformation → Generation → Optimization → Optimize by Generation \(YAGGA\)](#)
 - [5. Data Transformation → Attribute Set Reduction and Transformation → Principal Component Análisis](#)
 - [6. Data Transformation → Attribute Set Reduction and Transformation → Selection → Optimization → Optimize Selection](#)
 - [7. Data Transformation → Attribute Set Reduction and Transformation → Selection → Optimization → Optimize Selection \(Evolutionary\)](#)
 - [8. Data Transformation → Attribute Set Reduction and Transformation → Selection → Select Attributes](#)
 - [9. Data Transformation → Attribute Set Reduction and Transformation → Selection → Select by Weights](#)
 - [10. Data Transformation → Attribute Set Reduction and Transformation → Selection → Work on Subset](#)

- [11. Data Transformation → Attribute Set Reduction and Transformation → Transformation → Singular Value Decomposition](#)
- [12. Data Transformation → Data Cleansing → Replace Missing Values](#)
- [13. Data Transformation → Filtering → Filter Examples](#)
- [14. Data Transformation → Name and Role Modification → Rename](#)
- [15. Data Transformation → Name and Role Modification → Rename by Replacing](#)
- [16. Data Transformation → Name and Role Modification → Set Role](#)
- [17. Data Transformation → Set Operations → Append](#)
- [18. Data Transformation → Set Operations → Join](#)
- [19. Data Transformation → Sorting → Sort](#)
- [20. Data Transformation → Type Conversion → Discretization → Discretize by Frequency](#)
- [21. Data Transformation → Type Conversion → Discretization → Nominal to Binominal](#)
- [22. Data Transformation → Value Modification → Numerical Value Modification → Normalize](#)
- [23. Evaluation → Attributes → Performance \(Attribute Count\)](#)
- [24. Evaluation → Attributes → Performance \(CFS\)](#)
- [25. Evaluation → Performance Measurement → Classification and Regression → Performance \(Binominal Classification\)](#)
- [26. Evaluation → Performance Measurement → Classification and Regression → Performance \(Classification\)](#)
- [27. Evaluation → Performance Measurement → Classification and Regression → Performance \(Regression\)](#)
- [28. Evaluation → Performance Measurement → Performance](#)
- [29. Evaluation → Performance Measurement → Performance \(Min-Max\)](#)
- [30. Evaluation → Performance Measurement → Performance \(User-Based\)](#)
- [31. Evaluation → Significance → ANOVA](#)
- [32. Evaluation → Significance → T-Test](#)
- [33. Evaluation → Validation → Split Validation](#)
- [34. Evaluation → Validation → X-Validation](#)
- [35. Evaluation → Validation → Wrapper-X-Validation](#)
- [36. Export → Attributes → Write Constructions](#)
- [37. Export → Attributes → Write Weights](#)
- [38. Export → Other → Write Parameters](#)
- [39. Import → Attributes → Read Constructions](#)
- [40. Import → Attributes → Read Weights](#)
- [41. Import → Other → Read Parameters](#)
- [42. Modeling → Association and Item Set Mining → Create Association Rules](#)
- [43. Modeling → Association and Item Set Mining → FP-Growth](#)
- [44. Modeling → Attribute Weighting → Optimization → Optimize Weights \(Evolutionary\)](#)
- [45. Modeling → Attribute Weighting → Weight by Chi Squared Statistic](#)
- [46. Modeling → Classification and Regression → Bayesian Modeling → Naive Bayes](#)
- [47. Modeling → Classification and Regression → Function Fitting → Linear Regression](#)
- [48. Modeling → Classification and Regression → Lazy Modeling → k-NN](#)
- [49. Modeling → Classification and Regression → Meta Modeling → MetaCost](#)
- [50. Modeling → Classification and Regression → Meta Modeling → Stacking](#)
- [51. Modeling → Classification and Regression → Support Vector Modeling → Support Vector Machine](#)
- [52. Modeling → Classification and Regression → Support Vector Modeling → Support Vector Machine \(LibSVM\)](#)
- [53. Modeling → Classification and Regression → Tree Induction → Decision Tree](#)
- [54. Modeling → Clustering and Segmentation → k-Means](#)
- [55. Modeling → Model Application → Apply Model](#)

- [56. Modeling → Model Application → Group Models](#)
- [57. Modeling → Model Application → Thresholds → Apply Threshold](#)
- [58. Modeling → Model Application → Thresholds → Find Threshold](#)
- [59. Modeling → Model Application → Ungroup Models](#)
- [60. Process Control → Branch → Select Subprocess](#)
- [61. Process Control → Loop → Loop Attributes](#)
- [62. Process Control → Loop → Loop Values](#)
- [63. Process Control → Parameter → Optimize Parameters \(Grid\)](#)
- [64. Process Control → Multiply](#)
- [65. Process Control → Parameter → Set Parameters](#)
- [66. Repository Access → Retrieve](#)
- [67. Repository Access → Store](#)
- [68. Utility → Data Generation → Add Noise](#)
- [69. Utility → Data Generation → Generate Data](#)
- [70. Utility → Logging → Log](#)
- [71. Utility → Macros → Extract Macro](#)
- [72. Utility → Macros → Set Macro](#)
- [73. Utility → Miscellaneous → Free Memory](#)
- [74. Utility → Miscellaneous → Materialize Data](#)
- [75. Utility → Subprocess](#)
- [BIBLIOGRAFÍA](#)

Instalación

Web page de rapidminer: <http://rapid-i.com/index.php?lang=en>

Para descargar **rapidminer ce** (community edition) ir al siguiente link: <http://sourceforge.net/projects/yale/files/>

- Para Window\$: rapidminer-0.0.000x00-install.exe
- Para ambientes tipo Unix: rapidminer-0.0.000.zip

Para ver un manual online acerca de la instalación de radipminer seguir el siguiente enlace: <http://rapid-i.com/content/view/17/211/lang.en/>

Para ejecutar rapidminer hay que seguir básicamente los pasos a continuación:

- Abrir una consola o terminal.
- Posicionarse sobre el home de rapidminer.
- Ejecutar: **java -jar lib/rapidminer.jar**

Tutorial de RapidMiner 5.0

Este tutorial muestra los conceptos básicos de RapidMiner y las configuraciones de procesos simples que se pueden realizar. El usuario debe tener algún conocimiento en el dominio de minería de datos y ETL.

Siempre que este tutorial haga referencia al “Tutorial de RapidMiner”, significa la versión impresa disponible en <http://rapid-i.com>.

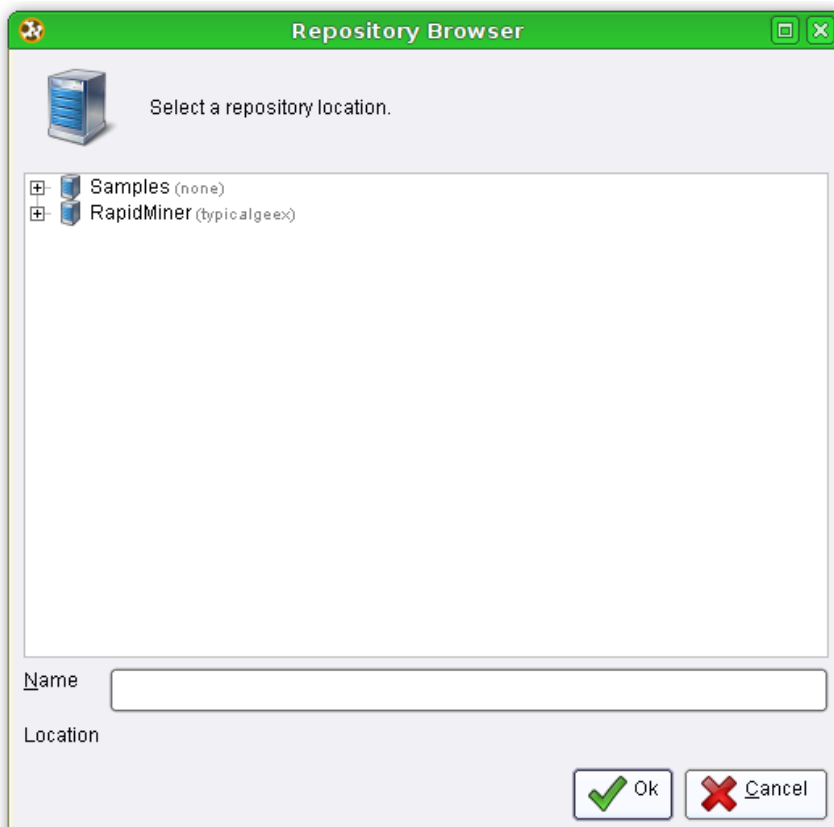
Es conveniente leer primero el capítulo del Tutorial de RapidMiner para una mejor motivación, pero también puede hacer el intento de comenzar con el tutorial en línea sin leer la versión impresa. Por favor lea los textos cuidadosamente e intente por lo menos los pasos sugeridos.

Por favor observe:

La mayor parte de RapidMiner proporciona información adicional si se detiene el puntero del ratón algunos instantes sobre el elemento (tool tip texts). De esta forma también se describen todos los operadores y parámetros. Al final de este tutorial se presenta un anexo con las descripciones de los operadores utilizados en el mismo y de los referenciados por éstos.

A continuación se presentarán una serie de ejemplos, cada uno de los cuales requiere que se cree un nuevo documento. Para ello se debe seleccionar en la barra de menú el icono .

Esto abrirá un nuevo documento y nos mostrará la siguiente ventana:



Aquí se puede seleccionar el lugar del repositorio en donde se guardará el documento, así como también el nombre que tendrá. O puede presionarse el botón “Cancel” para comenzar a trabajar sin guardar el documento momentáneamente.

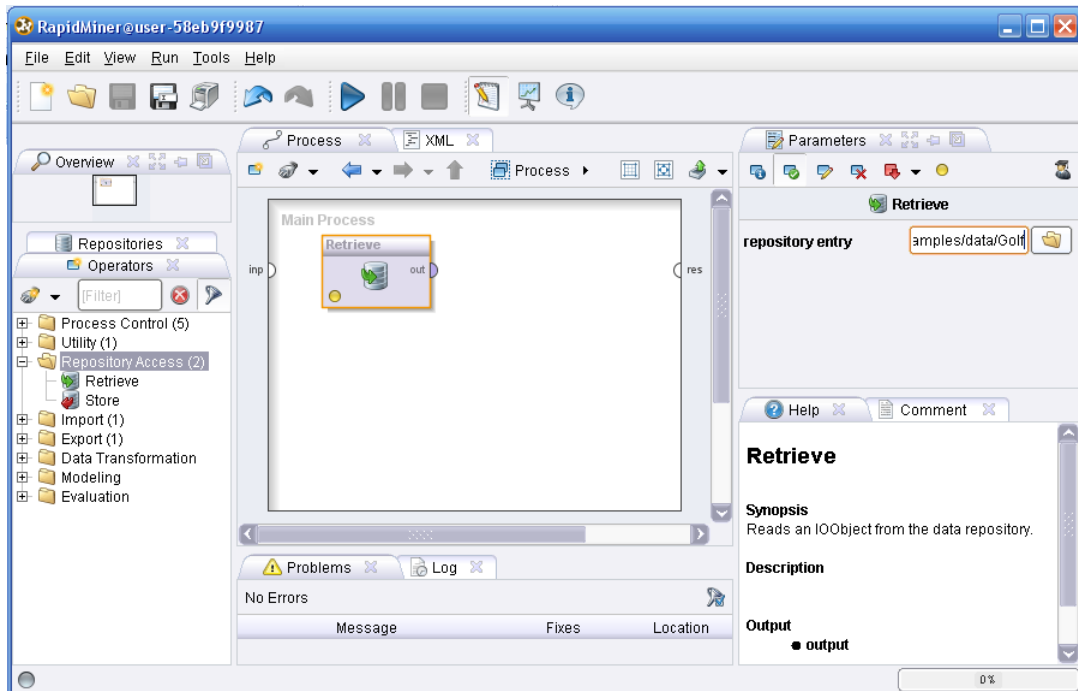
Ejemplo 1: Árbol de Decisión.

Este proceso comienza con la carga de datos. Después de finalizar el operador de entrada se realiza un típico paso de aprendizaje. Aquí se utiliza una implementación de un aprendiz de árbol de decisión que también puede manejar valores numéricos (similar al muy conocido algoritmo C4.5).

Cada operador puede requerir algunas entradas y entrega algunas salidas. Estos tipos de entrada y salida se pasan entre los operadores. En este ejemplo el primer operador "Input" no requiere ninguna entrada y entrega un conjunto de ejemplos como salida. Este conjunto de ejemplos es tomado por el aprendiz, el cual entrega la salida final: el modelo aprendido.

Debido a que este flujo de datos es lineal, el diseño del proceso se denomina “cadena de operadores”. Más adelante veremos procesos más sofisticados en la forma de árbol de operadores.

1. En el panel izquierdo seleccionar la pestaña “**Operators**”. Luego seleccionar el operador **Repository Access** → **Retrieve** y arrastrarlo a la zona de trabajo.
2. En la pestaña “**Parameters**” del panel derecho, utilizar el navegador a la derecha del parámetro *repository entry* para localizar el archivo //Samples/data/Golf.

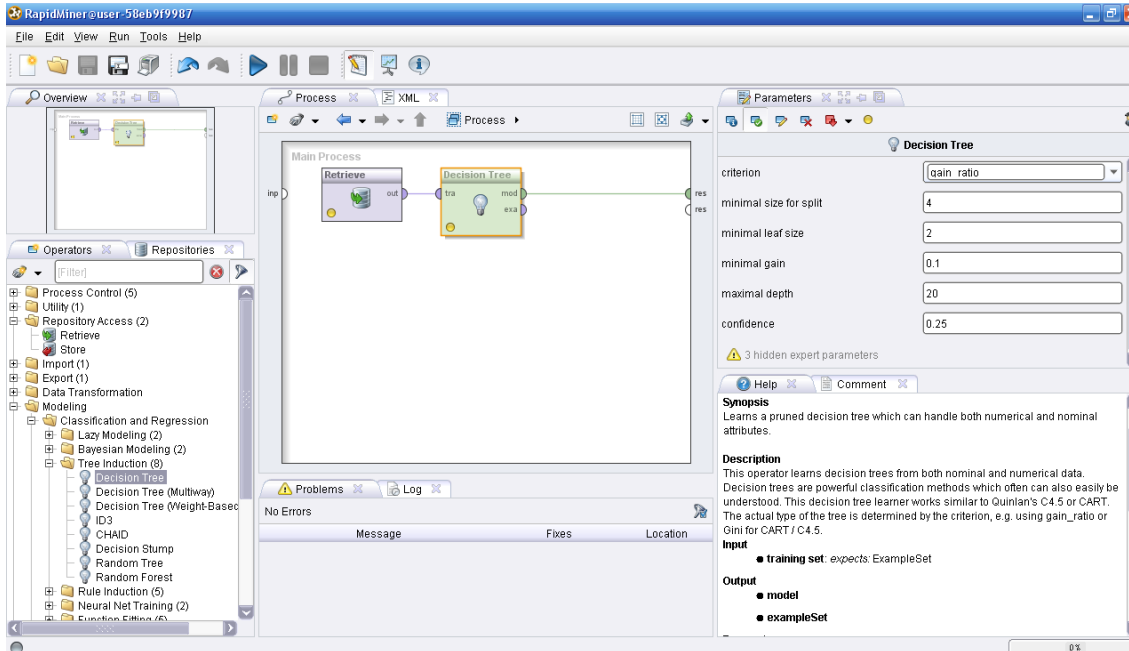



En esta imagen se muestran algunas de las vistas disponibles en RapidMiner. Para habilitar/deshabilitar las vistas, utilizar la entrada del menú **View** → **Show View** y para restaurar la perspectiva por defecto, seleccionar **View** → **Restore Default Perspective**.


3. En el panel izquierdo seleccionar el operador **Modeling** → **Classification and Regression** → **Tree Induction** → **Decision Tree** y arrastrarlo a la zona de trabajo.

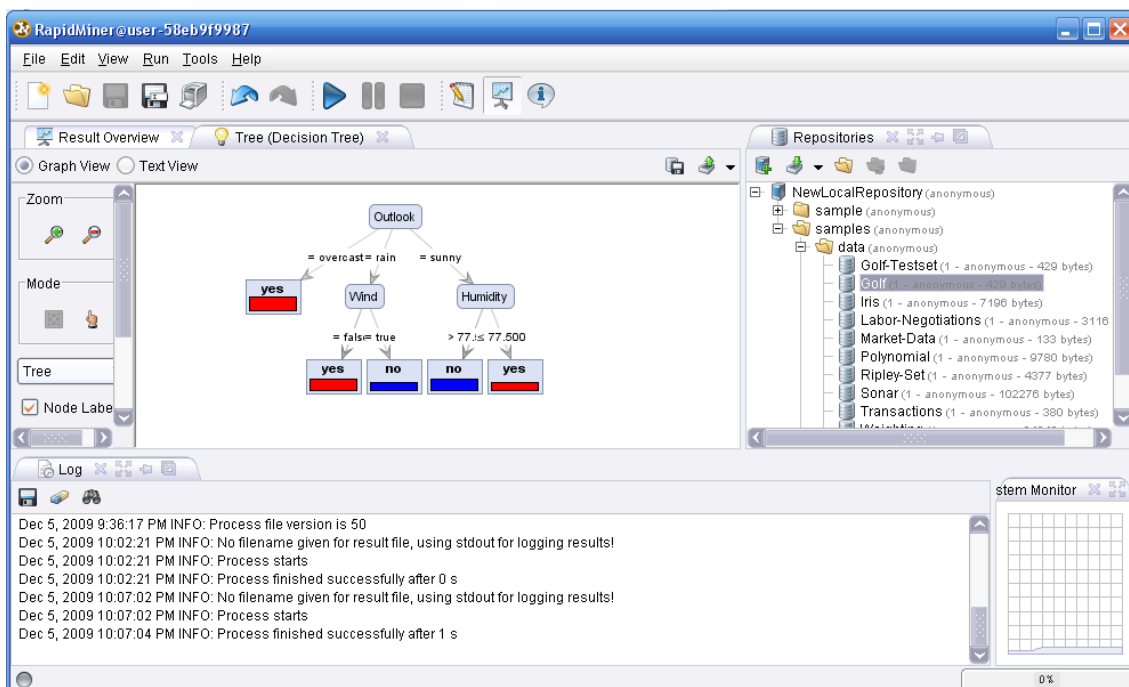
4. Conectar la salida del operador **Retrieve** a la entrada del operador **Decision Tree**, haciendo clic izquierdo en el conector **out** (output, salida) del primero y luego otro clic en el conector **tra** (training set, conjunto de entrenamiento) del segundo.

5. De la misma forma, conectar la salida **mod** (model, modelo) del operador **Decision Tree** al puerto **res** de la zona de trabajo.

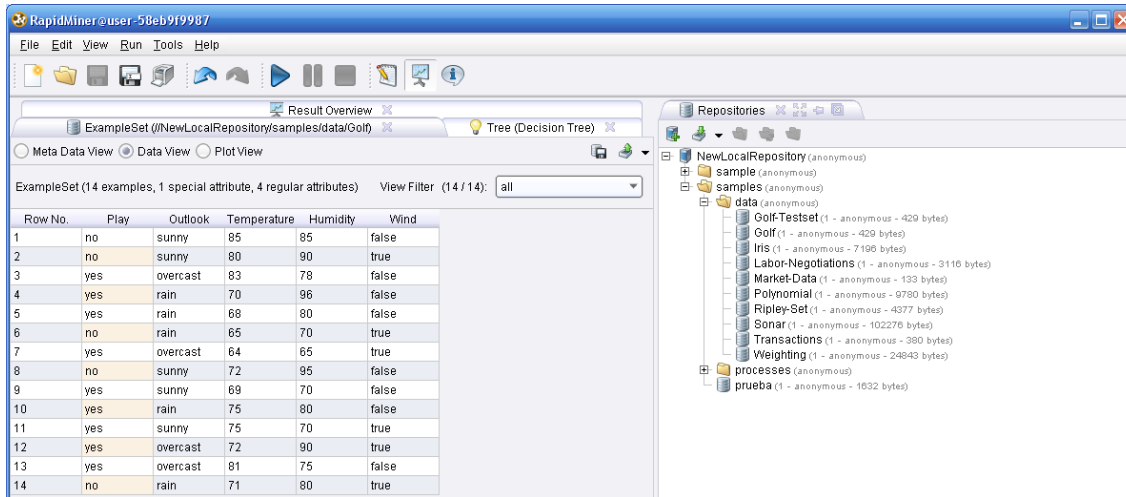


6. Presionar el icono “ejecutar”  en la barra de iconos de la parte superior del marco. El proceso debería comenzar y luego de un corto tiempo el visor de mensajes de la parte inferior del marco muestra el mensaje de que el proceso finalizó correctamente. El marco principal cambia a la vista de "Resultados", que muestra el árbol de decisión aprendido (una hipótesis que en RapidMiner se denomina Modelo).

7. Volver al modo edición ya sea por medio de la entrada del menú **View** → **Perspectives** → **Design**, el icono  de la barra de iconos, o presionando la tecla de función <F8>.



En este ejemplo se construyó un Modelo Predictivo para saber si se debería jugar o no al tenis, en base a los datos recogidos de experiencias anteriores. Para ver estos datos hacer doble clic sobre la tabla “Golf” de la pestaña “Repositories” de la derecha. Aparece otra pestaña entre las pestañas “Result Overview” y “Tree (Decision Tree)” de la vista de resultados, denominada “ExampleSet (//Samples/data/Golf)”. Seleccionar la opción *Data View*.



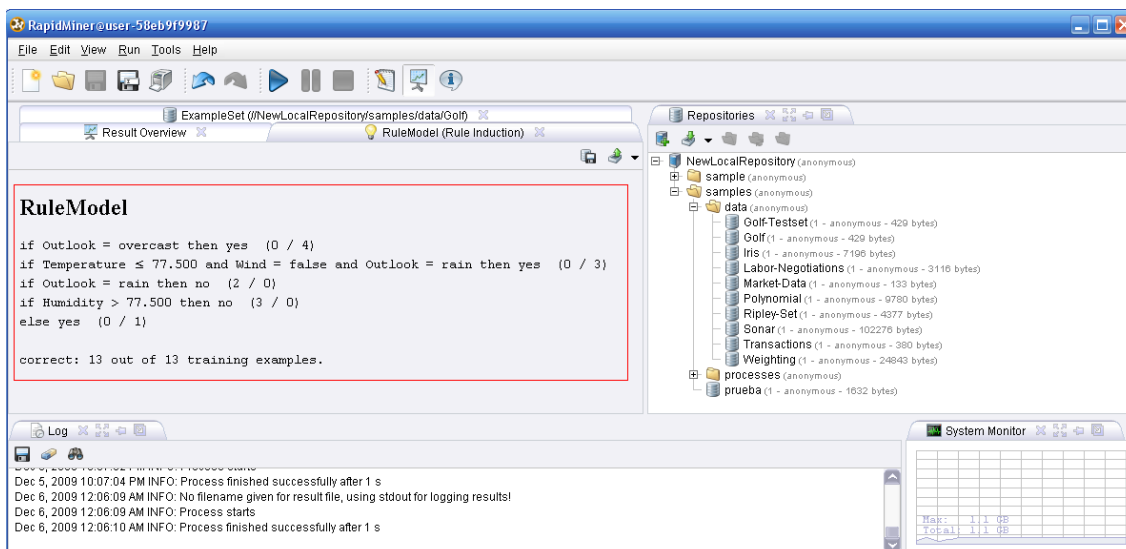
La primera columna es el Identificador de Casos, la segunda es el Atributo Objetivo y las restantes son los Atributos Predictores.

Ahora se puede utilizar este modelo para predecir si se debería jugar o no al tenis. Por ej., para la instancia: (Cielo = Soleado, Temperatura = 82, Humedad = 90, Ventoso = Verdadero) la respuesta es NO.

8. Reemplazar el aprendizaje por otro esquema de aprendizaje para tareas de clasificación. Hacer clic derecho sobre el operador **Decision Tree** y seleccionar **Replace Operator** → **Modeling** → **Classification and Regression** → **Rule Induction** → **Rule Induction**. Después de ejecutar el proceso cambiado con este ejemplo, se presenta el Nuevo modelo:

```

IF Cielo = Cubierto THEN Sí
IF Temperatura ≤ 77.500 AND Ventoso = Falso AND Cielo = Lluvioso THEN
Sí
IF Cielo = Lluvioso THEN No
IF Humedad > 77.500 THEN No ELSE Sí
    
```



Ejemplo 2: Reglas de Asociación.

Este proceso utiliza 2 importantes operadores de preprocesamiento: Primero el operador discretización de frecuencias, que discretiza atributos numéricos colocando los valores en intervalos de igual tamaño. Segundo, el operador filtro nominal a binominal crea para cada posible valor nominal de un atributo polinomial una nueva característica binominal (binaria) que es verdadera si el ejemplo tiene el valor nominal particular.

Estos operadores de preprocesamiento son necesarios debido a que determinados esquemas de aprendizaje no pueden manejar atributos de ciertos tipos de valores. Por ejemplo, el muy eficiente operador de minería de conjuntos de ítems frecuentes FPGrowth utilizado en esta configuración de proceso solo puede manejar características binominales y no numéricas ni polinominales.

El siguiente operador es el operador de minería de conjuntos de ítems frecuentes FPGrowth. Este operador calcula eficientemente conjuntos de valores de atributos que ocurren juntos con frecuencia. A partir de estos así llamados conjuntos de ítems frecuentes se calculan la mayoría de las reglas de confianza con el generador de reglas de asociación.

Nota: Para localizar más fácilmente un operador en el árbol de operadores, se puede escribir el nombre del mismo en el cuadro [Filter] de la pestaña “Operators”.

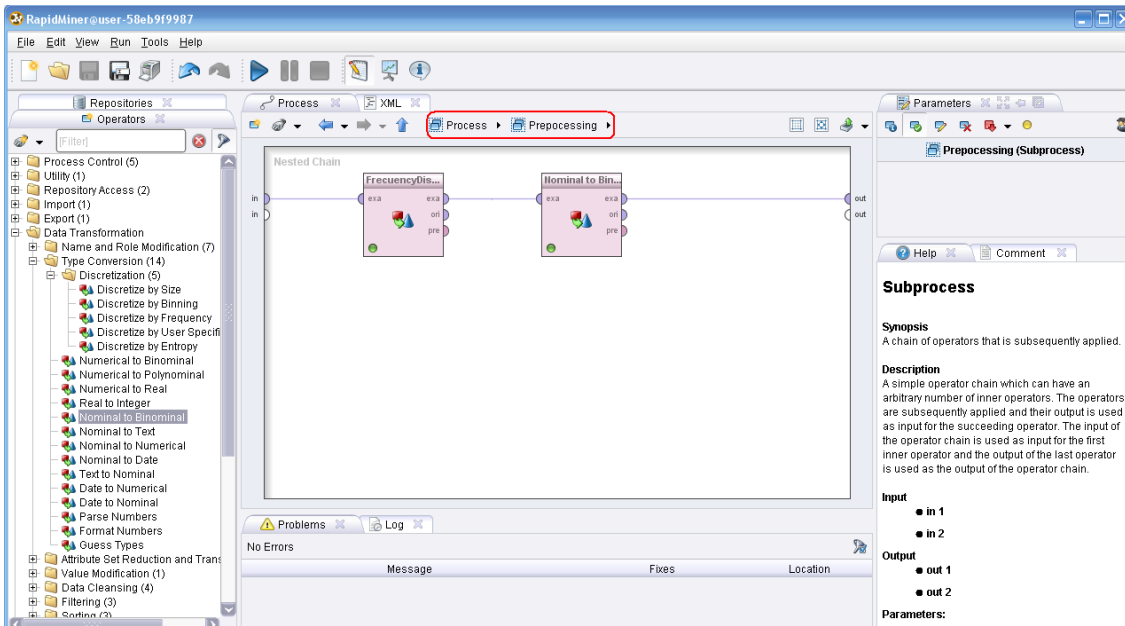
1. Agregar el operador **Retrieve** en la zona de trabajo y localizar el archivo //Samples/data/Iris con el navegador del parámetro *repository entry*.

2. Agregar el operador **Utility** → **Subprocess**. Cambiar el nombre del mismo a “Preprocesamiento” haciendo clic derecho y seleccionando “Rename” o bien presionando la tecla <F2>.

3. Conectar la salida del operador **Retrieve** a la entrada del operador **Preprocesamiento** (Subprocess) y luego doble clic sobre este último (observar que aparece un botón en la parte superior de este marco, al lado de “Process”, y que permite alternar entre el proceso y los subprocesses). En el panel **Nested Chain** del subnivel, agregar los siguientes operadores:

3.1 **Data Transformation** → **Type Conversion** → **Discretization** → **Discretize by Frequency**. Cambiar el nombre del mismo a “DiscretizaciónPorFrecuencias” y el parámetro *number of bins* (cantidad de intervalos) a 5, luego conectar la entrada **in** del panel a la entrada **exa** (example set, conjunto de ejemplos) de este operador.

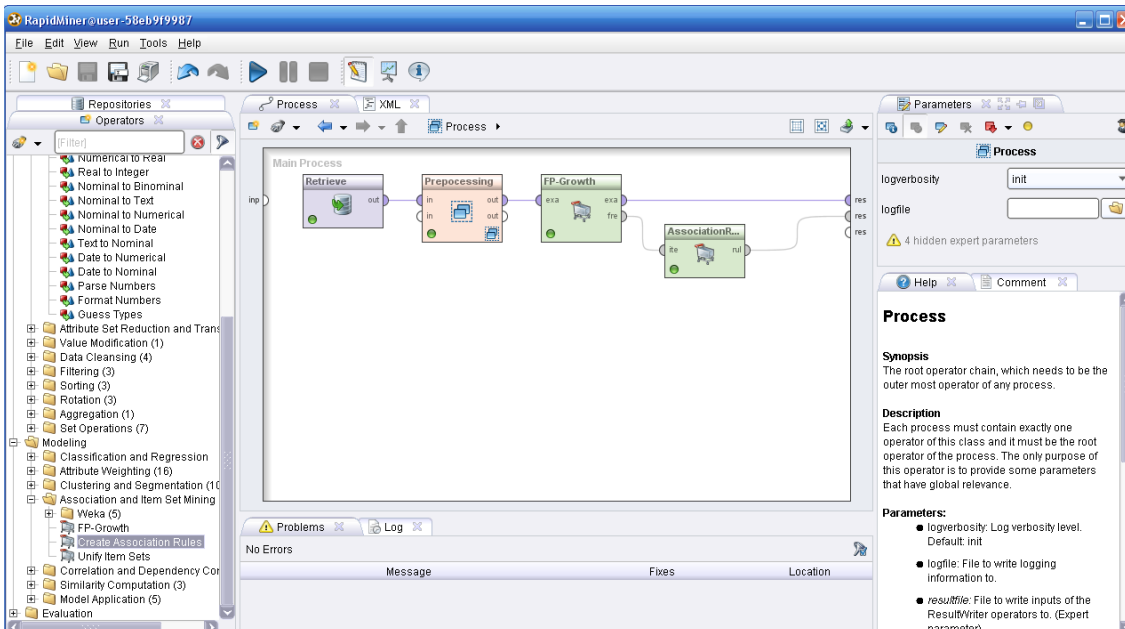
3.2 **Data Transformation** → **Type Conversion** → **Discretization** → **Nominal to Binominal**. Cambiar el nombre del mismo a “Nominal2Binominal”, conectar la salida **exa** del operador anterior a la entrada **exa** de este operador, y luego la salida **exa** de éste al conector **out** del panel.



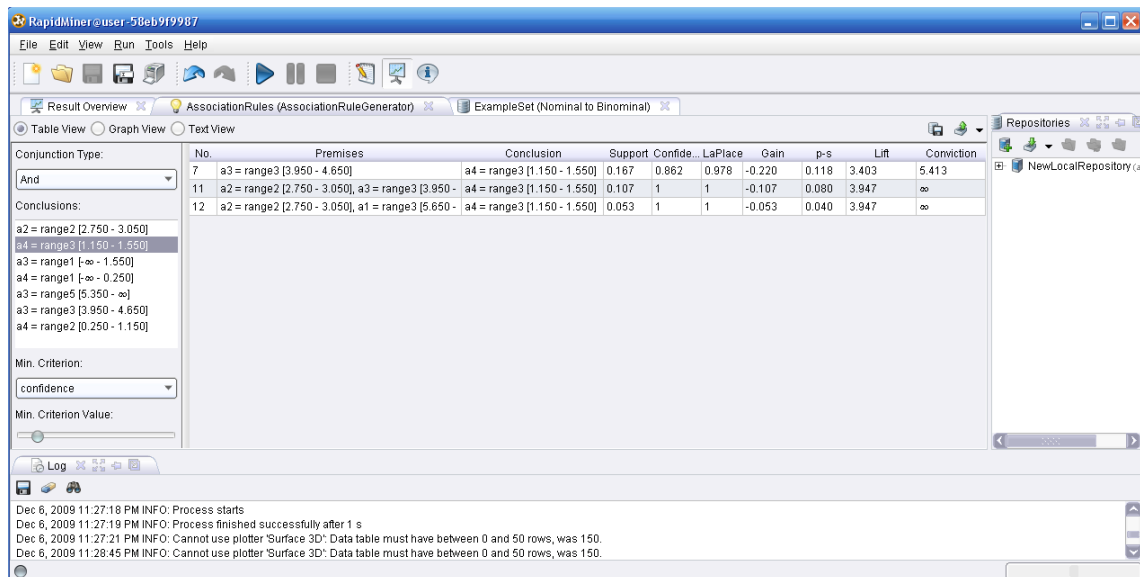
4. En el Proceso principal, agregar 2 operadores más:

4.1 **Modeling** → **Association and Item Set Mining** → **FP-Growth**. Cambiar el parámetro *min support* a 0.1, conectar la salida *out* del operador **Preprocesamiento** a la entrada *exa* de este operador y la salida *exa* de éste último al conector *res* (result, resultado) de la zona de trabajo.

4.2 **Modeling** → **Association and Item Set Mining** → **Create Association Rules**. Cambiar el nombre del mismo a “GeneradorReglasAsociación”, conectar la salida *fre* (frequent sets, conjuntos frecuentes) del operador **FP-Growth** a la entrada *ite* (item sets, conjuntos de elementos) de este operador y la salida *rul* (rules, reglas) de éste último a otro conector *res* de la zona de trabajo.



5. Ejecutar el proceso. El resultado se mostrará en un visor de reglas donde se puede seleccionar la conclusión deseada en una lista de selección en el lado izquierdo. Como para todas las otras tablas disponibles en RapidMiner, se pueden ordenar las columnas haciendo clic en la cabecera de la columna. Presionando CTRL durante estos clics permite la selección de hasta 3 columnas para ordenar.



Ejemplo 3: Stacking.

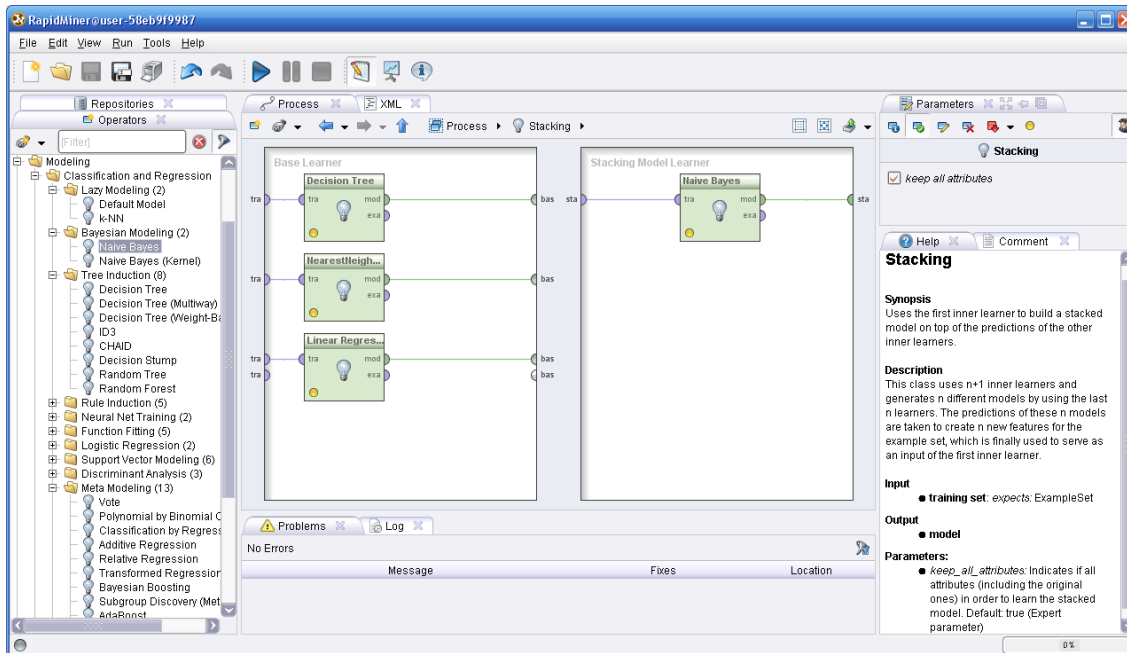
RapidMiner soporta Meta Aprendizaje incorporando uno o varios aprendices básicos como hijos en un operador de meta aprendizaje padre. En este ejemplo se genera un conjunto de datos con el operador ExampleSetGenerator y se aplica una versión mejorada de Stacking sobre este conjunto de datos. El operador Stacking contiene 4 operadores internos, el primero es un aprendiz que debe aprender el modelo stacked de las predicciones de los otros 4 operadores hijos (aprendices básicos). Otros esquemas de meta aprendizaje como Boosting o Bagging solo contienen un operador de aprendizaje interno. En ambos casos los parámetros de los esquemas de aprendizaje internos son establecidos directamente por los operadores de aprendizaje básicos. No es necesario tratar con los diferentes estilos de parámetros para los operadores internos y los de meta aprendizaje.

1. Agregar el operador **Utility** → **Data Generation** → **Generate Data** a la zona de trabajo. Cambiar el nombre del mismo a “GeneradorConjEjs”.
2. En la pestaña “Parameters” seleccionar “*simple polynomial classification*” de la lista desplegable del parámetro *target function* y cambiar el valor del parámetro *number examples* a 1000.
3. Agregar el operador **Modeling** → **Classification and Regression** → **Meta Modeling** → **Stacking** a la zona de trabajo.
4. Conectar la salida del operador **GeneradorConjEjs** (Generate Data) a la entrada **tra** del operador **Stacking** y la salida de éste último al conector **res** de la zona de trabajo.
5. Hacer doble clic sobre el operador **Stacking**. En el panel **Base Learner** del nivel inferior, agregar los siguientes operadores en paralelo, conectando la entrada y salida de los mismos a los puertos **tra** y **bas**, respectivamente:
 - 5.1 **Modeling** → **Classification and Regression** → **Tree Induction** → **Decision Tree**.
 - 5.2 **Modeling** → **Classification and Regression** → **Lazy Modeling** → **k-NN**. Cambiar el nombre del mismo a “VecinosCercanos” y el parámetro *k* a 5.
 - 5.3 **Modeling** → **Classification and Regression** → **Function Fitting** → **Linear Regression**.

En el panel **Stacking Model Learner** de la derecha, agregar el siguiente operador:

5.4 **Modeling** → **Classification and Regression** → **Bayesian Modeling** → **Naive Bayes**. Conectar la entrada y salida del mismo a los puertos **sta** izquierdo y derecho del panel, respectivamente.

6. Ejecutar el proceso y observar el resultado.



Ejemplo 4: K-Medias.

En muchos casos, no se puede definir ningún atributo objetivo (etiqueta) y los datos deben ser agrupados automáticamente. Este procedimiento se denomina "Clustering". RapidMiner soporta un amplio rango de esquemas de clustering que se pueden utilizar de la misma forma que cualquier otro esquema de aprendizaje. Esto incluye la combinación con todos los operadores de preprocesamiento.

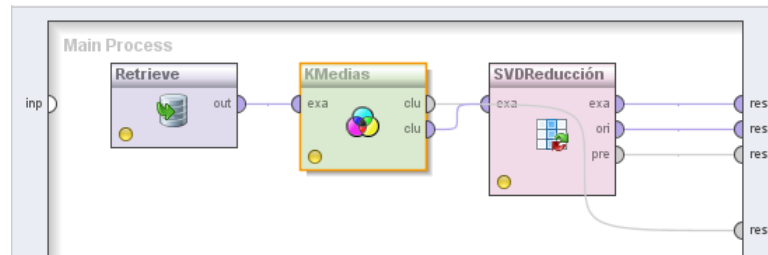
En este ejemplo, se carga el muy conocido conjunto de datos Iris (la etiqueta también se carga, pero sólo se utiliza para visualización y comparación y no para construir los clusters). Uno de los esquemas más simples de clustering, denominado KMeans, se aplica luego a este conjunto de datos. Después se realiza una reducción de dimensionalidad para que soporte mejor la visualización del conjunto de datos en 2 dimensiones.

Sólo realiza el proceso y compara el resultado del clustering con la etiqueta original (por ej., en la vista gráfica del conjunto de ejemplos). También se puede visualizar el modelo de cluster.

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Iris con el navegador del parámetro *repository entry*.

2. Agregar el operador **Modeling** → **Clustering and Segmentation** → **k-Means**. Cambiar el nombre del mismo a "KMedias" y el parámetro *k* a 3. Conectar la salida del operador **Retrieve** a la entrada **exa** de este operador y la salida **clu** (cluster model) de éste último al conector **res** del panel.

3. Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Transformation** → **Singular Value Decomposition**. Cambiar el nombre del mismo a “SVDReducción”. Conectar la salida **clu** (clustered set) del operador **KMedias** (k-Means) a la entrada **exa** de este operador y los 3 puertos de salida de éste último, **exa** (example set output), **ori** (original) y **pre** (preprocessing model), a conectores **res** del panel.



4. Ejecutar el proceso y observar el resultado.

Cluster Model

Cluster 0: 62 items
 Cluster 1: 38 items
 Cluster 2: 50 items
 Total number of items: 150

| Row No. | id | cluster | label | d0 | d1 |
|---------|-------|-----------|-------------|-------|--------|
| 1 | id_1 | cluster_2 | Iris-setosa | 0.062 | -0.130 |
| 2 | id_2 | cluster_2 | Iris-setosa | 0.058 | -0.111 |
| 3 | id_3 | cluster_2 | Iris-setosa | 0.057 | -0.118 |
| 4 | id_4 | cluster_2 | Iris-setosa | 0.057 | -0.106 |
| 5 | id_5 | cluster_2 | Iris-setosa | 0.061 | -0.131 |
| 6 | id_6 | cluster_2 | Iris-setosa | 0.068 | -0.131 |
| 7 | id_7 | cluster_2 | Iris-setosa | 0.057 | -0.117 |
| 8 | id_8 | cluster_2 | Iris-setosa | 0.061 | -0.121 |
| 9 | id_9 | cluster_2 | Iris-setosa | 0.054 | -0.100 |
| 10 | id_10 | cluster_2 | Iris-setosa | 0.059 | -0.112 |
| 11 | id_11 | cluster_2 | Iris-setosa | 0.065 | -0.137 |
| 12 | id_12 | cluster_2 | Iris-setosa | 0.060 | -0.114 |
| 13 | id_13 | cluster_2 | Iris-setosa | 0.057 | -0.112 |
| 14 | id_14 | cluster_2 | Iris-setosa | 0.052 | -0.116 |
| 15 | id_15 | cluster_2 | Iris-setosa | 0.068 | -0.165 |
| 16 | id_16 | cluster_2 | Iris-setosa | 0.071 | -0.159 |
| 17 | id_17 | cluster_2 | Iris-setosa | 0.065 | -0.147 |

Ejemplo 5: Visualización de SVM.

Este proceso muestra las posibilidades de visualización para las Máquinas de Vectores Soporte (SVM) y otros modelos de grandes márgenes basados en núcleos. El resultado de este proceso será un modelo de SVM para el cual se puede cambiar a la vista gráfica. Se proporcionan varias dimensiones para propósitos de graficación incluyendo las etiquetas del conjunto de entrenamiento, los valores alfa (multiplicadores de Lagrange), la información de si un ejemplo de entrenamiento es un vector soporte, los valores de la función (predicciones) para los ejemplos de entrenamiento y por supuesto los valores de los atributos para todos los

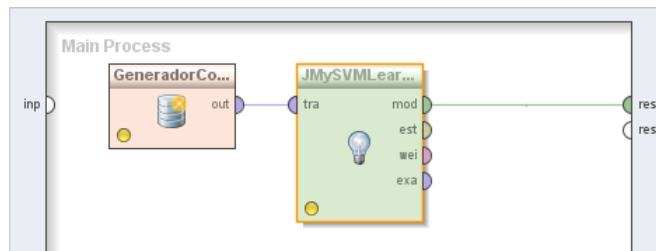
ejemplos de entrenamiento. Estos datos junto con el potente mecanismo graficador de RapidMiner permiten diferentes tipos de visualizaciones de SVM. Sólo pruebe alguna de ellas.

Sugerimos que por lo menos intente trazar los “valores de la función” contra los valores “alfa” en un diagrama de dispersión habitual. Esto puede darle una buena pista de si la función de núcleo utilizada es apropiada para el conjunto de datos. Lo mismo se aplica para trazar los cuartiles de los valores de la función y las alfas coloreadas por la etiqueta.

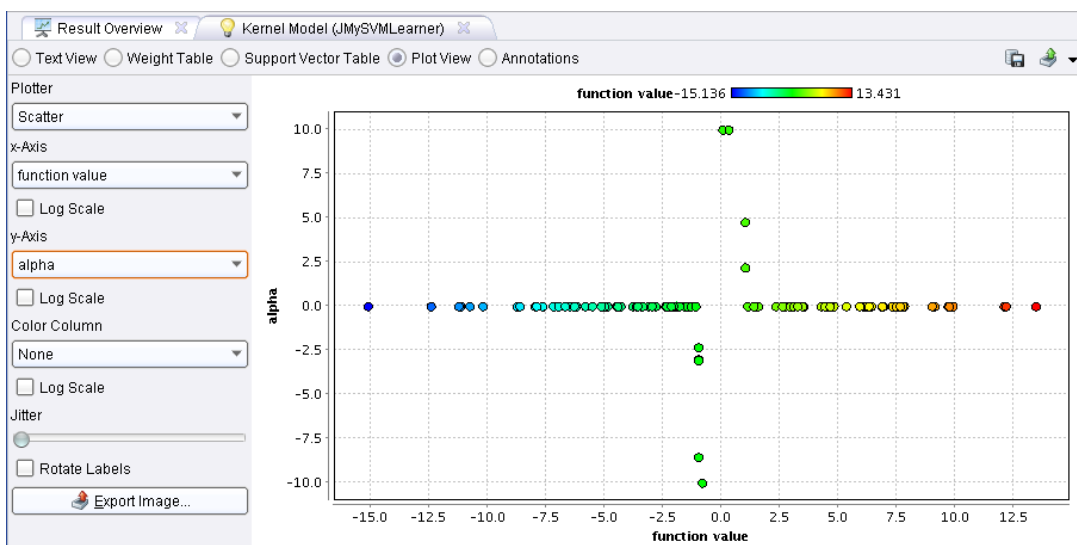
Una característica frecuentemente deseada es un diagrama coloreado de la densidad de los valores de la función. Esto se puede obtener en la vista gráfica de los modelos de SVM cambiando el graficador a “Densidad”, seleccionando dos atributos para los ejes x e y, “atributo1” y “atributo2” en este ejemplo, y estableciendo la “Densidad de Color” para la columna “valor de la función”. Esto conducirá al diagrama de densidad deseado. Si se configura el “Color de la Punta” a “vector soporte” o “alfa”, también obtendrá información sobre en qué puntos están los vectores soporte.

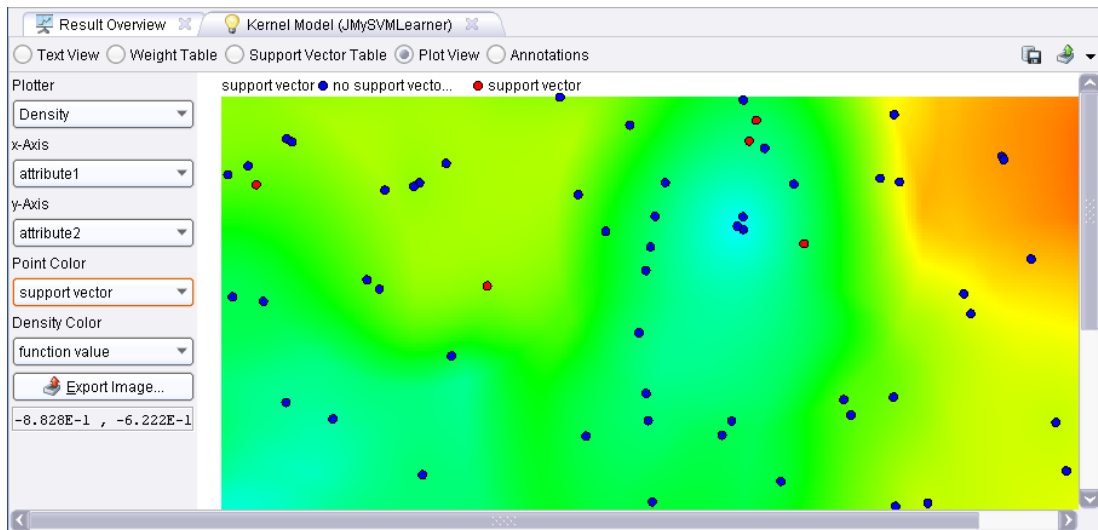
1. Agregar el operador **Utility** → **Data Generation** → **Generate Data** a la zona de trabajo. Cambiar el nombre del mismo a “GeneradorConjEjs” y los valores de los parámetros *target function* a “sum classification” (seleccionar de la lista desplegable), *number examples* a 200 y *number of attributes* a 2.

2. Agregar el operador **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine**. Cambiar el nombre del mismo a “AprendizJMySV” y el parámetro *C* a 10.0. Conectar la salida del operador **GeneradorConjEjs** (Generate Data) a la entrada **tra** de este operador y la salida **mod** (model) de éste último al puerto **res**.



3. Ejecutar el proceso y observar el resultado.






Ejemplo 6: Rellenado de valores faltantes.

Normalmente se emplea mucho tiempo de minería de datos para preprocesar los datos. RapidMiner ofrece varios operadores para leer datos de muchas fuentes diferentes y también operadores para procesar datos y facilitar el aprendizaje.

En muchas aplicaciones los datos contienen valores faltantes. Uno de los operadores de preprocesamiento disponibles los sustituye con el promedio / mín / máx del atributo. Otros operadores también pueden manejar valores infinitos.

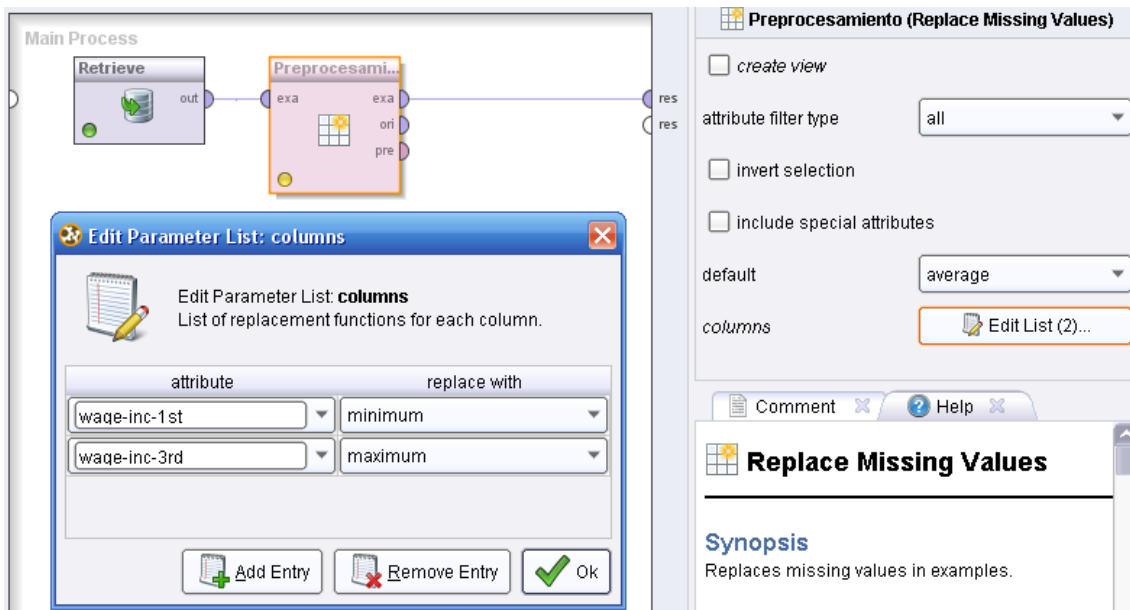
1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Labor-Negotiations con el navegador del parámetro *repository entry*.


2. Agregar el operador **Data Transformation** → **Data Cleansing** → **Replace Missing Values** a la zona de trabajo. Cambiar el nombre del mismo a “Preprocesamiento”. Conectar la salida del operador **Retrieve** a la entrada **exa** (example set input) del operador **Preprocesamiento** (Replace Missing Values) y la salida **exa** (example set output) de éste último al puerto **res**.

3. En el cuadro de la derecha (Parameters) activar la opción “Expert Mode” haciendo clic en el icono . Este modo también puede activarse/desactivarse presionando <F4> o en la barra de opciones superior con la opción “View → Expert Mode”.

El modo experto (Expert Mode) permite visualizar todos los atributos disponibles de cada operador.

4. Seleccionar el operador **Preprocesamiento** y hacer clic en el cuadro “**Edit List(0)...**” del parámetro *columns* para editar la lista de parámetros. En la ventana del editor pulsar el botón “Add Entry”. En la lista desplegable de la columna “attribute” especificar los atributos cuyos valores faltantes serán reemplazados: “wage-inc-1st” y “wage-inc-3rd”. En la lista desplegable de la columna “replace with” seleccionar la función que se utilizará para determinar el reemplazo de los valores faltantes de estos atributos: “minimum” y “maximum”, respectivamente.



4. Seleccionar el operador **Retrieve**. La pestaña “Parameters” de la derecha muestra los parámetros de este operador. El operador “Retrieve” sólo tiene el parámetro *repository entry*. Presionar <F7> o hacer clic derecho en este operador y luego se selecciona *Breakpoint After* (). Con esta acción se ha establecido un punto de interrupción, es decir, el proceso detendrá su ejecución después de este operador.

5. Ejecutar el proceso presionando el botón “Play” (<F11>). Como puede observarse el proceso comienza y se detiene después del punto de interrupción del operador “Retrieve”. En este momento RapidMiner muestra la salida del operador “Retrieve” en la pestaña ExampleSet (Retrieve). La columna “Missings” indica la cantidad de valores faltantes de un campo, por ej., el campo “pension” tiene 22 valores faltantes. Cambiar de *Meta Data View* a *Data View* para observar los valores faltantes. En la tabla de datos se pueden encontrar algunos signos de interrogación, que indican un valor faltante para una muestra (fila). El cuadro “View Filter” en la esquina superior derecha de la pestaña permite filtrar el conjunto de datos mediante ciertos criterios. Probar algunos filtros para ver qué muestras están completas y cuáles tienen valores faltantes.

6. Volver a la perspectiva de diseño (barra de menú: View/Perspectives/Design). Para sustituir los valores faltantes en los datos seleccionamos el operador **Preprocesamiento** (Replace Missing Values). Debemos asegurarnos que el modo experto este habilitado. El parámetro *attribute filter type* determina los atributos a los cuales se les aplicará el preprocesador. El parámetro *default* determina el valor con el que será reemplazado un valor faltante. Se pueden seleccionar varias opciones, por ej., el valor medio del atributo. Se pueden concatenar varios operadores de preprocesamiento para sustituir diferentes atributos con diferentes tipos de valores por defecto.

| Role | Name | Type | Statistics | Range | Missings |
|---------|--------------------------------|---------|---|---|----------|
| label | class | nominal | mode = good (26), least = bad (14) | bad (14), good (26) | 0 |
| regular | duration | integer | avg = 2.103 +/- 0.754 | [1.000 ; 3.000] | 1 |
| regular | wage-inc-1st | real | avg = 3.621 +/- 1.331 | [2.000 ; 6.900] | 1 |
| regular | wage-inc-2nd | real | avg = 3.913 +/- 1.281 | [2.000 ; 7.000] | 10 |
| regular | wage-inc-3rd | real | avg = 3.767 +/- 1.415 | [2.000 ; 5.100] | 28 |
| regular | col-adj | nominal | mode = none (14), least = tcf (4) | tcf (4), none (14), tc (6) | 16 |
| regular | working-hours | integer | avg = 37.811 +/- 2.717 | [27.000 ; 40.000] | 3 |
| regular | pension | nominal | mode = none (8), least = ret_allw (3) | none (8), empl_contr (7), ret_allw (3) | 22 |
| regular | standby-pay | integer | avg = 6.143 +/- 4.845 | [2.000 ; 13.000] | 33 |
| regular | shift-differential | integer | avg = 4.583 +/- 4.754 | [0.000 ; 25.000] | 16 |
| regular | education-allowance | nominal | mode = no (11), least = yes (7) | no (11), yes (7) | 22 |
| regular | statutory-holidays | integer | avg = 11.105 +/- 1.371 | [9.000 ; 15.000] | 2 |
| regular | vacation | nominal | mode = below-average (14), least = average (11) | generous (12), below-average (14), aver | 3 |
| regular | longterm-disability-assistance | nominal | mode = yes (11), least = no (5) | no (5), yes (11) | 24 |
| regular | contrib-to-dental-plan | nominal | mode = half (11), least = none (6) | none (6), half (11), full (8) | 15 |
| regular | bereavement-assistance | nominal | mode = yes (18), least = no (2) | no (2), yes (18) | 20 |
| regular | contrib-to-health-plan | nominal | mode = full (12), least = none (6) | none (6), half (6), full (12) | 16 |

| Row No. | class | duration | wage-inc-1st | wage-inc-2nd | wage-inc-3rd | col-adj | working-ho... | pension | standby-pay | shift-differe |
|---------|-------|----------|--------------|--------------|--------------|---------|---------------|------------|-------------|---------------|
| 1 | good | 1 | 5 | ? | ? | ? | 40 | ? | ? | 2 |
| 2 | good | 2 | 4.500 | 5.800 | ? | ? | 35 | ret_allw | ? | ? |
| 3 | good | ? | ? | ? | ? | ? | 38 | empl_contr | ? | 5 |
| 4 | good | 3 | 3.700 | 4 | 5 | tc | ? | ? | ? | ? |
| 5 | good | 3 | 4.500 | 4.500 | 5 | ? | 40 | ? | ? | ? |
| 6 | good | 2 | 2 | 2.500 | ? | ? | 35 | ? | ? | 6 |
| 7 | good | 3 | 4 | 5 | 5 | tc | ? | empl_contr | ? | ? |
| 8 | good | 3 | 6.900 | 4.800 | 2.300 | ? | 40 | ? | ? | 3 |
| 9 | good | 2 | 3 | 7 | ? | ? | 38 | ? | 12 | 25 |
| 10 | good | 1 | 5.700 | ? | ? | none | 40 | empl_contr | ? | 4 |
| 11 | good | 3 | 3.500 | 4 | 4.600 | none | 36 | ? | ? | 3 |
| 12 | good | 2 | 6.400 | 6.400 | ? | ? | 38 | ? | ? | 4 |
| 13 | bad | 2 | 3.500 | 4 | ? | none | 40 | ? | ? | 2 |
| 14 | good | 3 | 3.500 | 4 | 5.100 | tcf | 37 | ? | ? | 4 |
| 15 | good | 1 | 3 | ? | ? | none | 36 | ? | ? | 10 |

Ejemplo 7: Generador de ruido.

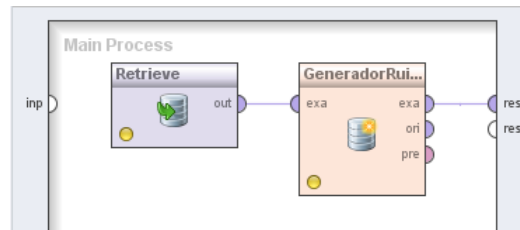
El NoiseOperator se puede utilizar para agregar ruido controlado o la característica de ruido al conjunto de datos. Esto es especialmente útil para estimar la performance de un preprocesamiento de características o la robustez de un aprendizaje específico.

RapidMiner también proporciona muchos otros operadores de preprocesamiento incluyendo un filtro de TFIDF, ofuscar, manejar series de valores y otros.

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Polynomial con el navegador del parámetro *repository entry*.

2. Agregar el operador **Utility** → **Data Generation** → **Add Noise** a la zona de trabajo. Cambiar el nombre del mismo a “GeneradorRuido” y los valores de los parámetros *random attributes* a 3 (cantidad de atributos aleatorios a agregar), *offset* a 5.0 (desplazamiento agregado a los valores de cada atributo aleatorio) y *linear factor* a 2.0 (factor lineal multiplicado por los valores de cada atributo aleatorio).

3. Conectar la salida del operador **Retrieve** a la entrada **exa** (example set input) del operador **GeneradorRuido** (Add Noise) y la salida **exa** (example set output) de éste último al puerto **res**.



4. Ejecutar el proceso y observar el resultado.

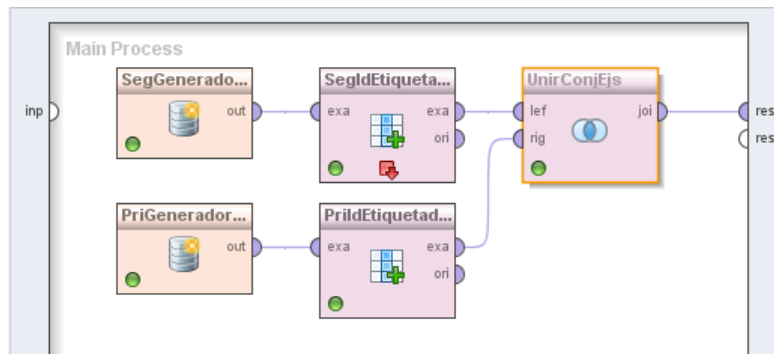
| Row No. | label | a1 | a2 | a3 | a4 | a5 | random | random1 | random2 |
|---------|---------|-------|-------|-------|-------|-------|--------|---------|---------|
| 1 | 84.149 | 0.637 | 8.442 | 2.116 | 2.772 | 8.448 | 0.831 | 1.949 | 7.531 |
| 2 | 113.443 | 4.595 | 4.388 | 4.926 | 2.682 | 8.618 | 9.047 | 4.636 | 3.986 |
| 3 | 47.259 | 4.292 | 0.861 | 9.158 | 6.607 | 4.085 | 8.477 | 6.628 | 5.664 |
| 4 | 32.091 | 8.560 | 3.856 | 1.038 | 1.688 | 2.893 | 5.067 | 5.854 | 2.555 |
| 5 | 64.152 | 2.272 | 1.834 | 6.301 | 1.938 | 2.070 | 2.865 | 5.556 | 1.957 |
| 6 | -8.582 | 3.956 | 0.344 | 4.337 | 7.563 | 8.591 | 4.374 | 2.557 | 7.855 |
| 7 | 263.017 | 2.220 | 7.843 | 6.463 | 8.732 | 9.282 | 5.523 | 3.480 | 3.036 |
| 8 | -2.617 | 6.105 | 1.075 | 1.163 | 9.241 | 7.239 | 4.769 | 5.357 | 6.888 |
| 9 | 163.178 | 2.691 | 3.572 | 7.806 | 6.843 | 3.054 | 4.891 | 2.231 | 3.741 |
| 10 | 107.792 | 4.263 | 7.369 | 0.821 | 6.688 | 3.976 | 5.125 | 1.741 | 6.603 |
| 11 | 293.574 | 1.364 | 9.049 | 8.860 | 2.573 | 1.538 | 5.958 | 2.737 | 6.045 |
| 12 | 145.880 | 6.226 | 8.381 | 0.351 | 3.639 | 6.234 | 5.920 | 6.608 | 4.400 |
| 13 | 225.355 | 9.812 | 3.942 | 5.305 | 6.661 | 7.118 | 5.662 | 4.246 | 5.114 |
| 14 | 5.180 | 1.383 | 1.469 | 7.947 | 2.043 | 0.030 | 2.732 | 5.053 | 2.971 |
| 15 | 126.321 | 0.697 | 9.944 | 0.001 | 4.557 | 3.258 | -1.241 | 4.471 | 6.534 |
| 16 | 219.180 | 3.163 | 8.397 | 7.110 | 3.342 | 6.405 | 2.360 | 7.030 | 3.320 |

Ejemplo 8: Unión de Conjuntos de Ejemplos.

El operador ExampleSetJoin de este proceso construye la unión de dos conjuntos dados de ejemplos. Observe que los atributos con nombres iguales serán renombrados durante el proceso de unión. Los conjuntos de ejemplos deben proporcionar un atributo de Id para determinar los ejemplos correspondientes. Después de alcanzar el punto de interrupción se pueden examinar los conjuntos de ejemplos de entrada. Después de reanudar el proceso, el resultado será el conjunto de ejemplos unidos.

1. Agregar 2 operadores **Utility** → **Data Generation** → **Generate Data** a la zona de trabajo. Cambiar el nombre de los mismos a “PriGeneradorConjEjs” y “SegGeneradorConjEjs”. Establecer el parámetro *target function* (función objetivo) de ambos en “sum classification” y el parámetro *number of attributes* (cantidad de atributos) en 5 y 10 respectivamente.
2. Agregar 2 operadores **Data Transformation** → **Attribute Set Reduction and Transformation** → **Generation** → **Generate ID**. Cambiar el nombre de los mismos a “PriIdEtiquetador” y “SegIdEtiquetador”.
3. Conectar las salidas de los operadores **PriGeneradorConjEjs** (Generate Data) y **SegGeneradorConjEjs** (Generate Data) a las entradas **exa** (example set input) de los operadores **PriIdEtiquetador** (Generate ID) y **SegIdEtiquetador** (Generate ID), respectivamente.

4. Agregar un operador **Data Transformation** → **Set Operations** → **Join**. Cambiar el nombre del mismo a “UnirConjEjs” y quitar la tilde del parámetro *remove double attributes* (visible en modo experto).
5. Conectar las salidas **exa** (example set output) de los operadores **PriGeneradorConjEjs** y **SegGeneradorConjEjs** a las entradas **rig** (right) y **lef** (left) del operador **UnirConjEjs** (Join), respectivamente, y la salida **joi** (join) de éste último al puerto **res**.
6. Seleccionar el operador **SegGeneradorConjEjs** y presionar F7 para establecer un punto de interrupción.
7. Ejecutar el proceso, observar el resultado parcial y reanudarlo luego de la interrupción presionando nuevamente el botón play, que ahora es de color verde.



Result Overview | ExampleSet (UnirConjEjs)

Meta Data View | **Data View** | Plot View | Annotations

ExampleSet (100 examples, 2 special attributes, 15 regular attributes) View Filter (100 / 100): all

| Row No. | id | label | att1 | att2 | att3 | att4 | att5 | att1_from_... | att2_from_... | att3_from_... |
|---------|----|----------|--------|--------|--------|--------|--------|---------------|---------------|---------------|
| 1 | 1 | negative | -7.880 | -1.058 | -9.287 | 7.909 | -8.624 | 2.468 | 7.267 | 1.292 |
| 2 | 2 | negative | 2.787 | -9.118 | -5.500 | -2.245 | 3.722 | -8.768 | 3.852 | -5.190 |
| 3 | 3 | positive | 6.330 | 8.760 | 7.846 | -2.853 | -2.518 | 5.304 | -2.017 | 0.911 |
| 4 | 4 | positive | -3.142 | -0.648 | -0.492 | 5.589 | 0.372 | -2.585 | -9.906 | 5.895 |
| 5 | 5 | negative | -9.961 | -8.048 | 9.541 | -3.590 | 8.651 | 9.677 | -7.499 | 3.662 |
| 6 | 6 | positive | -0.827 | 8.029 | 0.307 | 7.827 | 4.706 | -0.568 | 5.442 | -4.680 |
| 7 | 7 | negative | -0.337 | -8.219 | 2.468 | 9.925 | -3.922 | -0.258 | 7.311 | 3.584 |
| 8 | 8 | negative | -8.212 | -6.519 | 1.284 | -1.352 | 1.778 | 5.593 | -9.035 | -6.042 |
| 9 | 9 | negative | -3.550 | 3.006 | -1.893 | -0.337 | -2.414 | 0.629 | -7.551 | -4.398 |
| 10 | 10 | negative | -8.638 | -1.822 | 7.512 | 5.109 | -6.022 | -9.520 | 7.542 | 6.879 |
| 11 | 11 | negative | -0.712 | 4.869 | -6.699 | -6.980 | 6.484 | 9.391 | -7.647 | 2.415 |
| 12 | 12 | positive | -1.482 | 8.536 | -5.376 | 6.864 | -5.340 | 1.672 | 9.128 | 6.446 |
| 13 | 13 | negative | 1.874 | -1.365 | -3.848 | -7.341 | -8.813 | 1.150 | 0.374 | -9.564 |
| 14 | 14 | negative | 2.462 | 1.506 | -9.127 | -2.718 | 0.786 | 5.636 | -8.754 | -1.645 |
| 15 | 15 | positive | 8.178 | 4.569 | 3.112 | -1.088 | -8.627 | -7.297 | 0.434 | -9.544 |

Ejemplo 9: Validación Cruzada Numérica.

En muchos casos el modelo aprendido no es de interés sino la exactitud del modelo. Una posible solución para estimar la precisión del modelo aprendido es aplicarlo a datos de prueba etiquetados y calcular la cantidad de errores de predicción (u otros criterios de performance). Debido a que los datos etiquetados son poco frecuentes, a menudo se usan otros enfoques para estimar la performance de un esquema de aprendizaje. Este proceso muestra la “validación cruzada” en RapidMiner.

La validación cruzada divide los datos etiquetados en conjuntos de entrenamiento y de prueba. Los modelos se aprenden sobre los datos de entrenamiento y se aplican sobre los datos de prueba. Los errores de predicción se calculan y promedian para todos los subconjuntos. Este bloque de construcción se puede

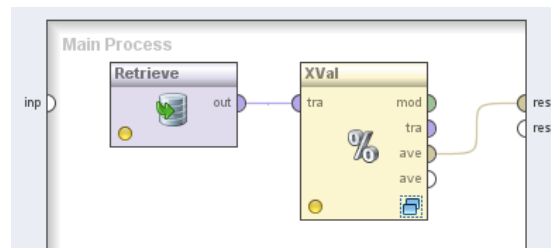
utilizar como operador interno para varios wrappers (contenedores) como los operadores de generación/selección de características.

Este es el primer ejemplo de un proceso más complejo. Los operadores construyen una estructura de árbol. Por ahora esto es suficiente para aceptar que el operador de validación cruzada requiere un conjunto de ejemplos como entrada y entrega un vector de valores de performance como salida. Además gestiona la división en ejemplos de entrenamiento y de prueba. Los ejemplos de entrenamiento se utilizan como entrada para el aprendiz de entrenamiento, el cual entrega un modelo. Este modelo y los ejemplos de prueba forman la entrada de la cadena de aplicadores que entregan la performance para este conjunto de prueba. Los resultados para todos los posibles conjuntos de prueba son recogidos por el operador de validación cruzada. Finalmente se calcula el promedio y se entrega como resultado.

Una de las cosas más difíciles para el principiante de RapidMiner es a menudo tener una idea del flujo de datos. La solución es sorprendentemente simple: el flujo de datos se asemeja a una búsqueda primero en profundidad a través de la estructura de árbol. Por ejemplo, después de procesar el conjunto de entrenamiento con el primer hijo de la validación cruzada del modelo aprendido, se entrega al segundo hijo (la cadena de aplicadores). Esta idea básica de flujo de datos es siempre la misma para todos los procesos y pensar en este flujo será muy conveniente para el usuario experimentado.

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Polynomial con el navegador del parámetro *repository entry*.

2. Agregar el operador **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “XVal” y el valor del parámetro *sampling type* (tipo de muestreo) a “*shuffled sampling*” (modo experto). Conectar la salida del operador **Retrieve** a la entrada **tra** (training) de este operador y la salida **ave** (averagable, promediable) de éste último al conector **res** (result) del panel.



3. Hacer doble clic sobre el operador **XVal** (X-Validation). En el panel **Training** del nivel inferior, agregar el siguiente operador:

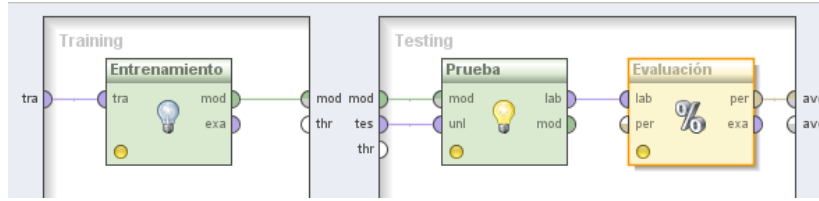
3.1 **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine (LibSVM)**. Cambiar el nombre del mismo a “Entrenamiento” y los valores de los parámetros *svm type* a “*epsilon-SVR*”, *kernel type* a “*poly*” y *C* a *1000.0*. Conectar la entrada **tra** (training) y salida **mod** (model) de este operador a los puertos **tra** y **mod** del panel, respectivamente

En el panel **Testing** de la derecha, agregar los siguientes operadores:

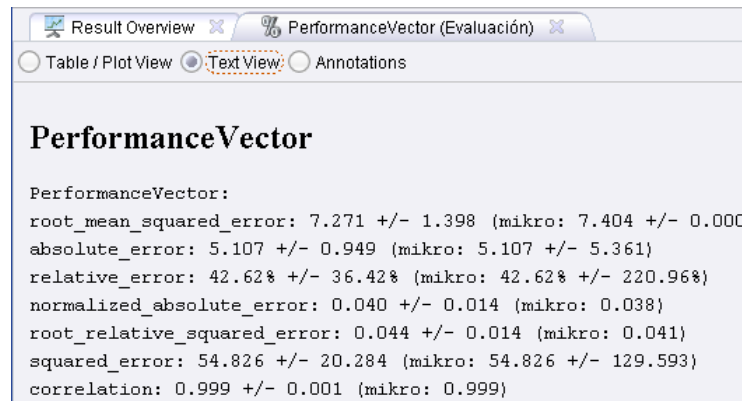
3.2 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “Prueba” y conectar las entradas **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

3.3 **Evaluation** → **Performance Measurement** → **Classification and Regression** → **Performance (Regression)**. Cambiar el nombre del mismo a “Evaluación” y tildar las siguientes opciones (además de las que están tildadas por defecto): *absolute error* (error absoluto), *relative error* (error relativo), *normalized absolute error* (error absoluto normalizado), *root relative squared error* (raíz cuadrada del error relativo al

cuadrado), *squared error* (error al cuadrado), y *correlation* (correlación). Conectar la salida **lab** (labeled data, datos etiquetados) del operador **Prueba** (Apply Model) a la entrada **lab** de este operador y la salida **per** (performance) de éste último al conector **ave** (averagable 1) del panel.



4. Ejecutar el proceso. El resultado es una estimación de la performance del esquema de aprendizaje sobre los datos de entrada.



5. Seleccionar el operador de evaluación y elegir otros criterios de performance. El criterio principal se utiliza para las comparaciones de performance, por ejemplo, en un wrapper.

6. Sustituir la validación cruzada **XVal** por otros esquemas de evaluación y ejecutar el proceso con ellos. Alternativamente, se puede verificar cómo funcionan otros aprendices sobre estos datos y sustituir el operador de entrenamiento.

Ejemplo 10: Aprendizaje sensitivo al costo y gráfico ROC.

Utilizamos los valores de confianza entregados por el aprendiz empleado en este proceso (predicciones flexibles en lugar de clasificaciones rígidas). Todos los aprendices de RapidMiner entregan estos valores de confianza, además de los valores pronosticados. Estos se pueden interpretar como una especie de garantía del aprendiz de que la predicción rígida (crisp) correspondiente es en realidad la etiqueta verdadera. En consecuencia, esto se denomina confianza.

En muchos escenarios de clasificación binaria, un error de predicción equivocada no ocasiona los mismos costos para ambas clases. Un sistema de aprendizaje debe tomar en cuenta estos costos asimétricos. Mediante el uso de las confianzas de predicción podemos convertir todos los aprendices de clasificación en aprendices sensibles al costo. Por lo tanto, ajustamos el umbral de confianza para hacer algunas predicciones (generalmente 0,5).

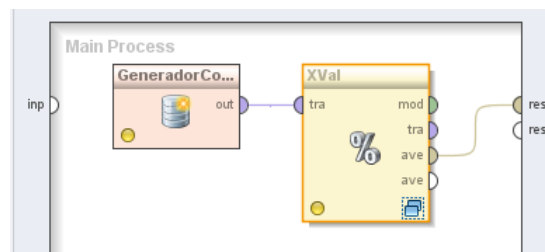
Un operador ThresholdFinder se puede utilizar para determinar el mejor umbral con respecto a los pesos de la clase. El siguiente operador ThresholdApplier mapea las predicciones flexibles (confianzas) a clasificaciones rígidas con respecto al valor del umbral determinado. El operador ThresholdFinder también puede producir una curva ROC para varios umbrales. Esta es una buena visualización de la performance de

un esquema de aprendizaje. El proceso se detiene cada vez que se grafica la curva ROC hasta que se pulsa el botón Ok (5 veces). El parámetro *show_ROC_plot* determina si el gráfico ROC se debe mostrar para todos.

Se puede encontrar información adicional sobre los operadores de validación utilizados en este proceso en el correspondiente directorio de ejemplos y, por supuesto, en la referencia de operadores del tutorial de RapidMiner.

1. Agregar el operador **Utility** → **Data Generation** → **Generate Data** a la zona de trabajo. Cambiar el nombre del mismo a “GeneradorConjEjs”. Establecer los parámetros *target function* (función objetivo) en “random dots classification”, *number examples* (cantidad de ejemplos) en 500, *number of attributes* (cantidad de atributos) en 2, *attributes lower bound* (límite inferior de los atributos) en 0.0 y *attributes upper bound* (límite superior de los atributos) en 25.0.

2. Agregar el operador **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “XVal” y el valor del parámetro *number of validations* (cantidad de validaciones) en 5. Conectar la salida del operador **GeneradorConjEjs** (Generate Data) a la entrada **tra** (training) de este operador y la salida **ave** (averagable, promediable) de este último al conector **res** (result) del panel.



3. Hacer doble clic sobre el operador **XVal** (X-Validation). En el panel **Training** del nivel inferior, agregar el siguiente operador:

3.1 **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine (LibSVM)**. Cambiar el nombre del mismo a “AprendizLibSVM” y el valor del parámetro *gamma* a 1.0. Conectar la entrada **tra** (training) y salida **mod** (model) de este operador a los puertos **tra** y **mod** del panel, respectivamente

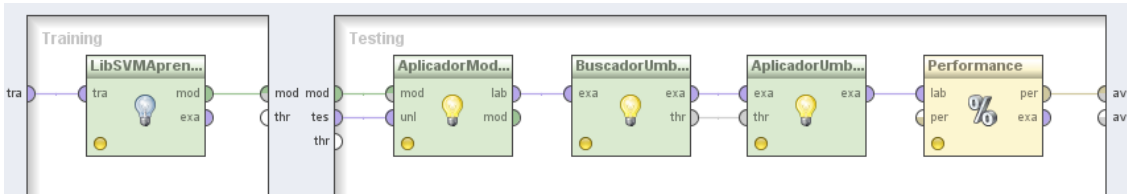
En el panel **Testing** de la derecha, agregar los siguientes operadores:

3.2 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModelo” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

3.3 **Modeling** → **Model Application** → **Thresholds** → **Find Threshold**. Cambiar el nombre del mismo a “BuscadorUmbral” y tildar la opción *show roc plot* (mostrar gráfico ROC), además de la opción *use example weights* (utilizar pesos de las muestras), que está seleccionada por defecto. Conectar la salida **lab** (labelled data, datos etiquetados) del operador **AplicadorModelo** (Apply Model) a la entrada **exa** (example set) de este operador.

3.4 **Modeling** → **Model Application** → **Thresholds** → **Apply Threshold**. Cambiar el nombre del mismo a “AplicadorUmbral”. Conectar las salidas **exa** (example set, conjunto de ejemplos) y **thr** (threshold, umbral) del operador **BuscadorUmbral** (Find Threshold) a la entradas **exa** y **thr** de este operador, respectivamente.

3.5 **Evaluation** → **Performance Measurement** → **Performance**. Conectar la salida **exa** (example set) del operador **AplicarModelo** a la entradas **lab** (labelled data) de este operador y la salida **per** de éste último al conector **ave** (averagable) del panel.



4. Ejecutar el proceso y observar el resultado.

Result Overview PerformanceVector (Performance)

Table / Plot View Text View Annotations

Criterion Selector

- accuracy
- precision
- recall
- AUC (optimistic)
- AUC
- AUC (pessimistic)

Multiclass Classification Performance Annotations

Table View Plot View

accuracy: 93.20% +/- 3.82% (mikro: 93.20%)

| | true negative | true positive | class precision |
|----------------|---------------|---------------|-----------------|
| pred. negative | 424 | 0 | 100.00% |
| pred. positive | 34 | 42 | 55.26% |
| class recall | 92.58% | 100.00% | |

Result Overview PerformanceVector (Performance)

Table / Plot View **Text View** Annotations

PerformanceVector

PerformanceVector:
 accuracy: 93.20% +/- 3.82% (mikro: 93.20%)
 ConfusionMatrix:
 True: negative positive
 negative: 424 0
 positive: 34 42
 precision: 58.20% +/- 11.37% (mikro: 55.26%) (positive class: positive)
 ConfusionMatrix:
 True: negative positive
 negative: 424 0
 positive: 34 42
 recall: 100.00% +/- 0.00% (mikro: 100.00%) (positive class: positive)
 ConfusionMatrix:
 True: negative positive
 negative: 424 0
 positive: 34 42
 AUC (optimistic): 0.981 +/- 0.010 (mikro: 0.981) (positive class: positive)
 AUC: 0.981 +/- 0.010 (mikro: 0.981) (positive class: positive)
 AUC (pessimistic): 0.981 +/- 0.010 (mikro: 0.981) (positive class: positive)

Ejemplo 11: Aprendizaje de Costos Asimétricos.

Este proceso muestra cómo se puede obtener un umbral de un clasificador soft (flexible) y aplicarlo a un conjunto independiente de prueba.

El aprendiz utilizado en este proceso realiza predicciones flexibles (soft) en lugar de clasificaciones rígidas (crisp). Las confianzas de predicción entregadas por todos los aprendices de RapidMiner que pueden manejar etiquetas nominales (clasificación) serán utilizadas como predicciones flexibles.

El operador **ThresholdFinder** se utiliza para determinar el mejor umbral con respecto a los pesos de la clase. En este caso, una clasificación errónea de la primera clase (negativo) tendrá un costo 5 veces mayor que el otro error.

Observe que se debe ejecutar un operador **ModelApplier** sobre el conjunto de prueba antes de que se pueda encontrar un umbral. Debido a que este modelo debe ser aplicado de nuevo más tarde, el aplicador del modelo guarda el modelo de entrada.

El **IOConsumer** asegura que la predicción se realiza sobre el conjunto de datos correcto.

Los últimos pasos aplican el modelo y el umbral sobre el conjunto de datos en cuestión.

1. Agregar el operador **Utility** → **Data Generation** → **Generate Data** a la zona de trabajo. Cambiar el nombre del mismo a “GeneradorConjEntren” y los parámetros *target function* a “polynomial classification” y *number of attributes* a 20.

2. Agregar el operador **Modeling** → **Classification and Regression** → **Lazy Modeling** → **k-NN**. Cambiar el nombre del mismo a “VecinosMásCercanos” y el parámetro *k* a 10. Conectar la salida del operador **GeneradorConjEntren** (Generate Data) a la entrada **tra** (training set) de este operador.

3. Agregar otro operador **Utility** → **Data Generation** → **Generate Data**. Cambiar el nombre del mismo a “GeneradorConjPrueba” y los valores de los parámetros *target function* a “polynomial classification” y *number of attributes* en 20.

4. Agregar el operador **Modeling** → **Model Application** → **Apply Model** y cambiar el nombre del mismo a “PruebaModelo”. Conectar la salida **mod** del operador **VecinosMásCercanos** (k-NN) y la salida del operador **GeneradorConjPrueba** (Generate Data) a las entradas **mod** (model) y **unl** (unlabelled data) de este operador, respectivamente.

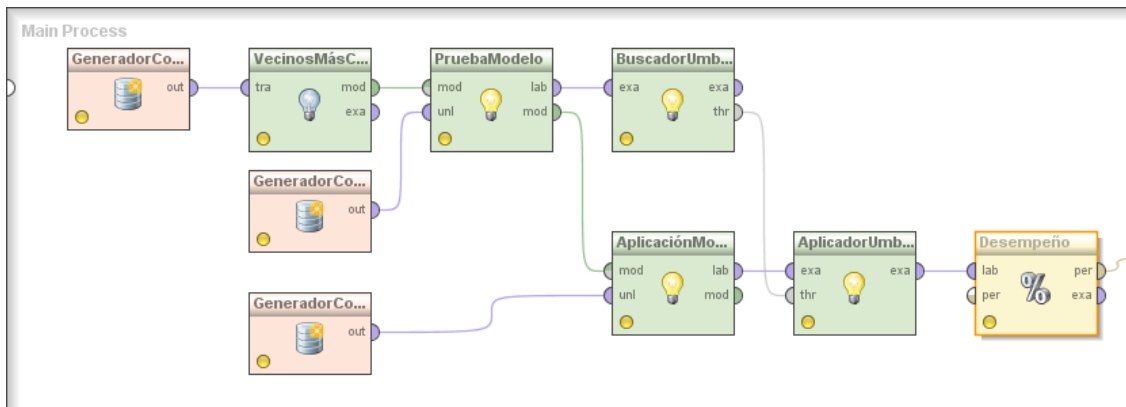
5. Agregar el operador **Modeling** → **Model Application** → **Thresholds** → **Find Threshold**. Cambiar el nombre del mismo a “BuscadorUmbral” y el parámetro *misclassification costs second* a 2.0. Conectar la salida **lab** (labelled data) del operador **PruebaModelo** (Apply Model) a la entrada **exa** (example set) de este operador.

6. Agregar otro operador **Utility** → **Data Generation** → **Generate Data**. Cambiar el nombre del mismo a “GeneradorConjAplic” y los valores de los parámetros *target function* a “polynomial classification”, *number examples* a 200 y *number of attributes* a 20.

7. Agregar otro operador **Modeling** → **Model Application** → **Apply Model** y cambiar el nombre del mismo a “AplicaciónModelo”. Conectar la salida **mod** del operador **PruebaModelo** y la salida del operador **GeneradorConjAplic** (Generate Data) a las entradas **mod** y **unl** de este operador, respectivamente.

8. Agregar el operador **Modeling** → **Model Application** → **Thresholds** → **Apply Threshold** y cambiar el nombre del mismo a “AplicadorUmbral”. Conectar la salida **thr** del operador **BuscadorUmbral** (Find Threshold) y la salida **lab** del operador **AplicaciónModelo** (Apply Model) a las entradas **thr** y **exa** de este operador, respectivamente.

9. Agregar el operador **Evaluation** → **Performance Measurement** → **Performance**. Conectar la salida **exa** (example set) del operador **AplicadorUmbral** (Apply Threshold) a las entradas **lab** de este operador y la salida **per** de éste último al conector **res** del panel.



Result Overview PerformanceVector (Desempeño)

Table / Plot View Text View Annotations

Criterion Selector

- accuracy
- precision
- recall
- AUC (optimistic)
- AUC
- AUC (pessimistic)

Multiclass Classification Performance Annotations

Table View Plot View

accuracy: 74.00%

| | true negative | true positive | class precision |
|----------------|---------------|---------------|-----------------|
| pred. negative | 86 | 35 | 71.07% |
| pred. positive | 17 | 62 | 78.48% |
| class recall | 83.50% | 63.92% | |

PerformanceVector:

accuracy: 74.00%

ConfusionMatrix:

True: negative positive

negative: 86 35

positive: 17 62

precision: 78.48% (positive class: positive)

ConfusionMatrix:

True: negative positive

negative: 86 35

positive: 17 62

recall: 63.92% (positive class: positive)

ConfusionMatrix:

True: negative positive

negative: 86 35

positive: 17 62

AUC (optimistic): 0.843 (positive class: positive)

AUC: 0.797 (positive class: positive)

AUC (pessimistic): 0.750 (positive class: positive)

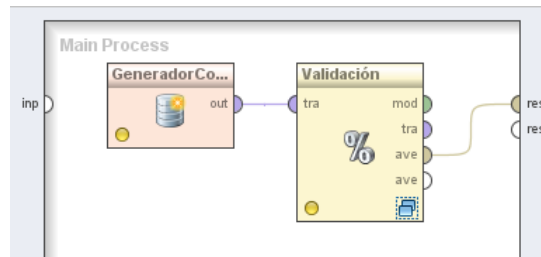
Ejemplo 12: Aprendizaje Sensible al Costo.

Este proceso es otro ejemplo de aprendizaje sensible al costo, es decir, para el caso donde diferentes errores de predicción causarían diferentes costos. Además del operador de preprocesamiento ThresholdFinder, que también es capaz de entregar gráficos ROC para 2 clases, hay otro operador que se puede utilizar para aprendizaje sensible al costo.

Este operador es parte del aprendizaje, grupo Meta, y se denomina MetaCost. Se utiliza como cualquier otro esquema de meta-aprendizaje y debe contener otro operador de aprendizaje interno, en este caso se utiliza el aprendizaje de árbol de decisión.

La matriz de costos utilizada para el aprendizaje sensible al costo se puede definir mediante el editor de matrices (en el operador MetaCost, presionar “*Edit Matrix...*” del parámetro *cost matrix*). El formato básico del parámetro *cost matrix* es [K11 K12 ... K1m, K21 K22 ... K2m, ... ; Kn1 ... Knm], por ejemplo, para una matriz de costos de 2x2 de un problema de clasificación binaria es [0 1 ; 10 0]. Este ejemplo quiere decir que tanto los costos para los errores de predicción de la primera clase como para los de la segunda son 10 veces más altos que los otros tipos de errores.

1. Agregar el operador **Utility** → **Data Generation** → **Generate Data**. Cambiar el nombre del mismo a “GeneradorConjEjs” y los valores de los parámetros *target function* a “polynomial classification” y *number examples* a 300.
2. Agregar el operador **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “Validación” y el valor del parámetro *number of validations* a 5. Conectar la salida del operador **GeneradorConjEjs** (Generate Data) a la entrada **tra** (training) de este operador y la salida **ave** (averagable 1) de este último al conector **res** del panel.



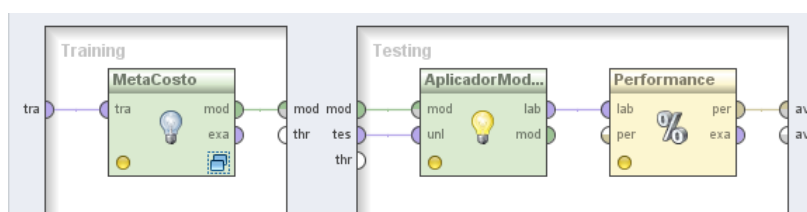
3. Hacer doble clic sobre el operador **Validación** (X-Validation). En el panel **Training** del nivel inferior, agregar el siguiente operador:

3.1 **Modeling** → **Classification and Regression** → **Meta Modeling** → **MetaCost**. Cambiar el nombre del mismo a “MetaCosto” y utilizar el editor de matrices del parámetro *cost matrix* para ingresar los valores de la matriz de costos [0 1; 10 0]., Conectar la entrada **tra** y salida **mod** de este operador a los puertos **tra** y **mod** del panel, respectivamente.

En el panel **Testing** de la derecha, agregar los siguientes operadores:

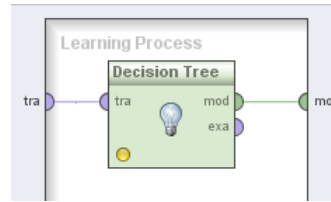
3.2 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModelo” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

3.3 Agregar el operador **Evaluation** → **Performance Measurement** → **Performance**. Conectar la salida **lab** del operador **AplicadorModelo** (Apply Model) a la entradas **lab** de este operador y la salida **per** de éste último al conector **ave** del panel.



4. Hacer doble clic sobre el operador **MetaCosto** (MetaCost). En el panel **Learning Process** del nivel inferior, agregar el siguiente operador:

4.1 **Modeling** → **Classification and Regression** → **Tree Induction** → **Decision Tree**. Conectar la entrada **tra** y la salida **mod** del mismo a los puertos **tra** y **mod** del panel, respectivamente.



Resultado:

```
PerformanceVector:
accuracy: 83.33% +/- 6.58% (mikro: 83.33%)
ConfusionMatrix:
True:  negative positive
negative: 145    49
positive: 1     105
precision: 99.20% +/- 1.60% (mikro: 99.06%) (positive class: positive)
ConfusionMatrix:
True:  negative positive
negative: 145    49
positive: 1     105
recall: 68.28% +/- 13.00% (mikro: 68.18%) (positive class: positive)
ConfusionMatrix:
True:  negative positive
negative: 145    49
positive: 1     105
AUC (optimistic): 0.985 +/- 0.005 (mikro: 0.985) (positive class: positive)
AUC: 0.978 +/- 0.013 (mikro: 0.978) (positive class: positive)
AUC (pessimistic): 0.976 +/- 0.014 (mikro: 0.976) (positive class: positive)
```

Ejemplo 13: Análisis de Componentes Principales.

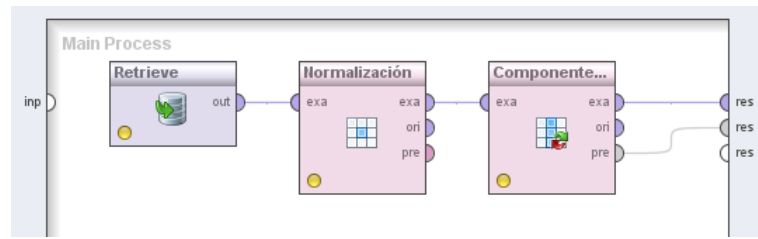
El cálculo de componentes principales se usa con frecuencia como un paso de procesamiento de la transformación de características. Puede reducir la dimensionalidad del conjunto de datos en cuestión, mientras se preservan las varianzas más importantes de los datos. Ejecutar el proceso y comprobar la salida en la vista gráfica del conjunto de datos Iris cargado y transformado por este proceso.

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Iris con el navegador del parámetro *repository entry*.

2. Agregar el operador **Data Transformation** → **Value Modification** → **Numerical Value Modification** → **Normalize**. Cambiar el nombre del mismo a “Normalización” y conectar la salida del operador **Retrieve** a la entrada **exa** (example set input) de este operador.

3. Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Principal Component Analysis**. Cambiar el nombre del mismo a “Componentes Principales” y conectar la

salida **exa** del operador **Normalización** (Normalize) a la entrada **exa** de este operador, y las salidas **exa** y **pre** a sendos conectores **res** del panel.



Resultados:

Result Overview | PCA (ComponentesPrincipales) | ExampleSet (ComponentesPrincipales)

Eigenvalues Eigenvectors Cumulative Variance Plot Annotations

| Component | Standard Deviation | Proportion of Variance | Cumulative Variance |
|-----------|--------------------|------------------------|---------------------|
| PC 1 | 1.706 | 0.728 | 0.728 |
| PC 2 | 0.960 | 0.230 | 0.958 |
| PC 3 | 0.384 | 0.037 | 0.995 |
| PC 4 | 0.144 | 0.005 | 1.000 |

Result Overview | PCA (ComponentesPrincipales) | ExampleSet (ComponentesPrincipales)

Eigenvalues Eigenvectors Cumulative Variance Plot Annotations

| Attribute | PC 1 | PC 2 | PC 3 | PC 4 |
|-----------|--------|--------|--------|--------|
| a1 | 0.522 | -0.372 | 0.721 | 0.262 |
| a2 | -0.263 | -0.926 | -0.242 | -0.124 |
| a3 | 0.581 | -0.021 | -0.141 | -0.801 |
| a4 | 0.566 | -0.065 | -0.634 | 0.524 |

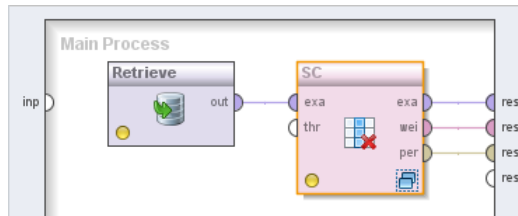
Ejemplo 14: Selección Forward.

Las transformaciones del espacio de atributos pueden facilitar el aprendizaje de manera que simples esquemas de aprendizaje puedan ser capaces de aprender funciones complejas. Esta es la idea básica de la función kernel. Pero incluso sin esquemas de aprendizaje basados en kernel, la transformación del espacio de características, puede ser necesaria para alcanzar buenos resultados de aprendizaje.

RapidMiner ofrece varios métodos diferentes de selección, construcción, y extracción de características. Este proceso de selección (la muy conocida selección forward) utiliza una validación cruzada interna para la estimación de la performance. Este elemento sirve como evaluación de la aptitud para todos los conjuntos candidatos de características. Debido a que se toma en cuenta la performance de un determinado esquema de aprendizaje, nos referimos a los procesos de este tipo como “enfoques wrapper”.

Además, el operador log del proceso grafica los resultados intermedios.

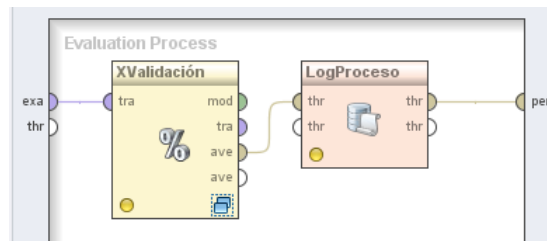
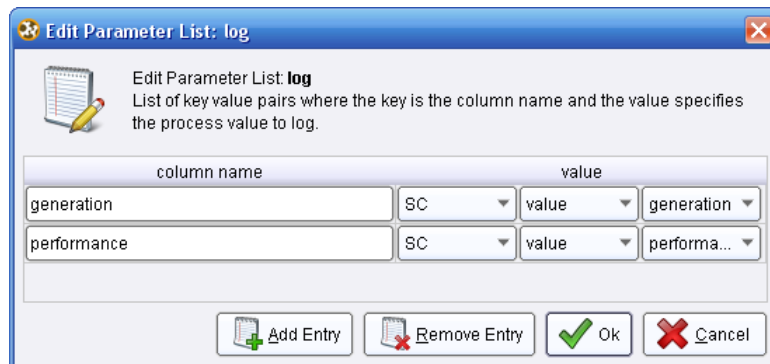
1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Polynomial con el navegador del parámetro *repository entry*.
2. Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Selection** → **Optimization** → **Optimize Selection**. Cambiar el nombre del mismo a “SC” y conectar la salida del operador **Retrieve** a la entrada **exa** de este operador, y las salidas **exa** (example set out), **wei** (weights) y **per** (performance) a conectores **res** del panel.



3. Hacer doble clic sobre el operador **SC** (Optimize Selection). En el panel **Evaluation Process** del nivel inferior, agregar los siguientes operadores:

3.1 **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “XValidación” y el parámetro *sampling type* a “shuffled sampling”. Conectar la entrada **exa** del panel a la entrada **tra** (training) de este operador.

3.2 **Utility** → **Logging** → **Log**. Cambiar el nombre del mismo a “LogProceso” y conectar la salida **ave** (averagable 1) del operador **XValidación** (X-Validation) a la entrada **thr** (through 1) de este operador y la salida (through 1) del mismo, al conector **per** (performance) del panel. En el parámetro *log* de este operador editar la lista de parámetros para incluir los campos “generation” y “performance”:



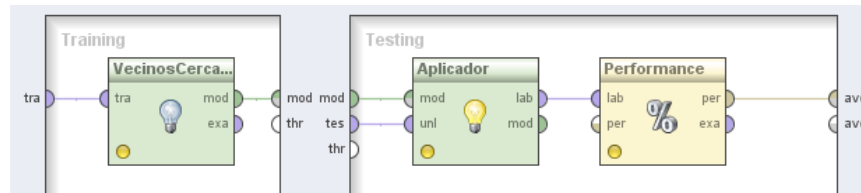
4. Hacer doble clic sobre el operador **Validación** (X-Validation). En el panel **Training** del nivel inferior, agregar el siguiente operador:

4.1 **Modeling** → **Classification and Regression** → **Lazy Modeling** → **k-NN**. Cambiar el nombre del mismo a “VecinosCercanos” y el parámetro *k* a 5. Conectar la entrada **tra** y salida **mod** de este operador a los puertos **tra** y **mod** del panel, respectivamente

En el panel **Testing** de la derecha, agregar los siguientes operadores:

4.2 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “Aplicador” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

4.3 **Evaluation** → **Performance Measurement** → **Performance**. Conectar la salida **lab** del operador **Aplicador** (Apply Model) a la entradas **lab** de este operador y la salida **per** de éste último al conector **ave** del panel.



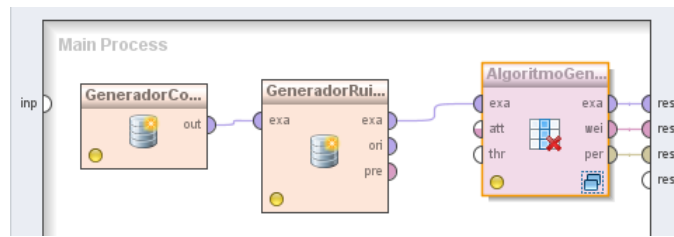
5. Ejecutar el proceso y cambiar en la vista “Result”, seleccionar la pestaña “Log”. Graficar “performance” contra “generation” del operador de selección de características.
6. Seleccionar el operador de selección de características en el panel del proceso principal. Cambiar el parámetro *selection direction* de “forward” (selección hacia adelante) a “backward” (eliminación hacia atrás). Reiniciar el proceso. Todas las características serán seleccionadas.
7. Seleccionar el operador de selección de características. Hacer clic derecho para abrir el menú contextual y reemplazar el operador por otro esquema de selección de características (por ejemplo un algoritmo genético).
8. Observar la lista del operador de registro del proceso. Cada vez que se aplica recoge los datos especificados. Consultar el Tutorial RapidMiner para más explicaciones. Después de cambiar el operador de selección de características al enfoque de algoritmos genéticos, hay que especificar los valores correctos. Utilizar el operador de registro de proceso para registrar los valores en línea.

Ejemplo 15: Selección Multiobjetivos.

Este es otro enfoque muy simple de selección genética de características. Debido a otro esquema de selección, el operador de selección de características no sólo intenta maximizar la performance entregada por el evaluador del conjunto de características, sino que también intenta minimizar el número de características. El resultado es un gráfico de Pareto diagramado durante la optimización.

Después de finalizada la optimización, el usuario puede hacer doble clic en las soluciones óptimas de Pareto y ver qué conjunto de características está representado por un punto. El gráfico de Pareto no sólo brinda una mejor comprensión de la cantidad total de características necesarias, sino también la compensación entre la cantidad de características y el rendimiento, y un ranking de características.

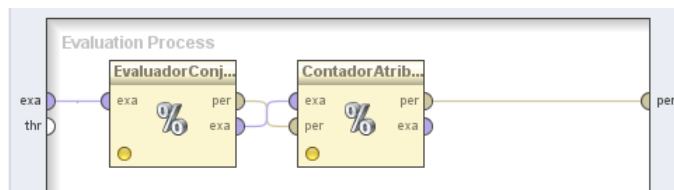
1. Agregar el operador **Utility** → **Data Generation** → **Generate Data**. Cambiar el nombre del mismo a “GeneradorConjEjs” y los valores de los parámetros *target function* a “sum classification”, *number examples* a 200, y *number of attributes* a 10.
2. Agregar el operador **Utility** → **Data Generation** → **Add Noise**. Cambiar el nombre del mismo a “GeneradorRuido” y los valores de los parámetros *random attributes* a 10 y *label noise* a 0.0. Conectar la salida del operador **GeneradorConjEjs** (Generate Data) a la entrada *exa* de este operador.
3. Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Selection** → **Optimization** → **Optimize Selection (Evolutionary)**. Cambiar el nombre del mismo a “AlgoritmoGenético” y los valores de los parámetros *population size* (tamaño de la población) a 30 y *maximum number of generations* a 15. Conectar la salida *exa* del operador **GeneradorRuido** (Add Noise) a la entrada *exa* de este operador, y las salidas *exa* (example set out), *wei* (weights) y *per* (performance) a conectores *res* del panel.



4. Hacer doble clic sobre el operador **AlgoritmoGenético** (Optimize Selection (Evolutionary)). En el panel **Evaluation Process** del nivel inferior, agregar los siguientes operadores:

4.1 **Evaluation** → **Attributes** → **Performance (CFS)**. Cambiar el nombre del mismo a “EvaluadorConjCaractsCFS” y conectar la entrada **exa** del panel a la entrada **exa** de este operador.

4.2 **Evaluation** → **Attributes** → **Performance (Attribute Count)** y cambiar el nombre del mismo a “ContadorAtributos”. Conectar las salidas **per** y **exa** del operador **EvaluadorConjCaractsCFS** (Performance (CFS)) a las entradas **per** y **exa** de este operador, respectivamente, y la salida **per** de éste último al conector **per** del panel.



Ejemplo 16: Validación Wrapper.

Así como en el aprendizaje, también es posible que ocurra overfitting (sobreajuste) durante el preprocesamiento. Para estimar la performance de generalización de un método de preprocesamiento, RapidMiner soporta varios operadores de validación para los pasos de preprocesamiento. La idea básica es la misma que para todos los otros operadores de validación con una ligera diferencia: el primer operador interno debe producir un conjunto de ejemplos transformado, el segundo debe producir un modelo de ese conjunto de datos transformado y el tercer operador debe producir un vector de performance de ese modelo sobre un conjunto de prueba apartado y transformado de la misma forma.

Este es un proceso más complejo que muestra la capacidad de RapidMiner para construir procesos a partir de elementos ya conocidos. En este proceso, se utiliza una variante especial de un operador de validación cruzada para estimar la performance de una transformación del espacio de características, es decir, la simple selección de características forward en este caso.

El bloque de construcción completo de selección de características es ahora el primer operador interno de un WrapperXValidation que, al igual que la validación cruzada normal, utiliza un subconjunto para la transformación del espacio de características y el aprendizaje basado en el conjunto de características determinado. Una segunda cadena de aplicadores se utiliza para estimar la performance sobre un conjunto de pruebas que no fue utilizado para el aprendizaje y la selección de características. La performance estimada y un vector de pesos de atributos se devuelven como resultado.

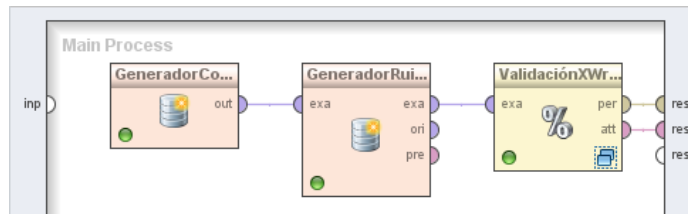
Observe el MinMaxWrapper después del evaluador de performance interno. Este operador encapsula los criterios de performance dados de tal manera que no sólo los valores medios, sino también los valores mínimos se calculan durante la validación cruzada. Arbitrariamente las combinaciones lineales ponderadas de las medias mínima y normal conducen a mejorar la capacidad de generalización. Sólo cambiar el

parámetro *weighting* (ponderación) a $0,0$ o desactivar el operador en el menú contextual o eliminarlo del proceso para ver el efecto. La performance disminuye rápidamente cuando se utiliza solamente la performance media como criterio de selección.

1. Agregar el operador **Utility** → **Data Generation** → **Generate Data**. Cambiar el nombre del mismo a “GeneradorConjEjs” y los valores de los parámetros *target function* a “sum”, *number examples* a 60, y *number of attributes* a 3.

2. Agregar el operador **Utility** → **Data Generation** → **Add Noise**. Cambiar el nombre del mismo a “GeneradorRuido” y el valor del parámetro *random attributes* a 3. Conectar la salida del operador **GeneradorConjEjs** (Generate Data) a la entrada **exa** de este operador.

3. Agregar el operador **Evaluation** → **Validation** → **Wrapper-X-Validation** y cambiar el nombre del mismo a “ValidaciónXWrapper”. Conectar la salida **exa** del operador **GeneradorRuido** (Add Noise) a la entrada **exa** de este operador, y las salidas **per** (performance vector out) y **att** (attribute weights out) de éste último a los conectores **res** del panel.



4. Hacer doble clic sobre el operador **ValidaciónXWrapper** (Wrapper-X-Validation). En el panel **Attribute Weighting** del nivel inferior, agregar el siguiente operador:

4.1 Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Selection** → **Optimization** → **Optimize Selection**. Cambiar el nombre del mismo a “SelecciónCaracterísticas”. Conectar la entrada **wei** (weighting set source) del panel a la entrada **exa** de este operador y la salida **wei** (weights) del mismo al conector **att** (attribute weights sink) del panel.

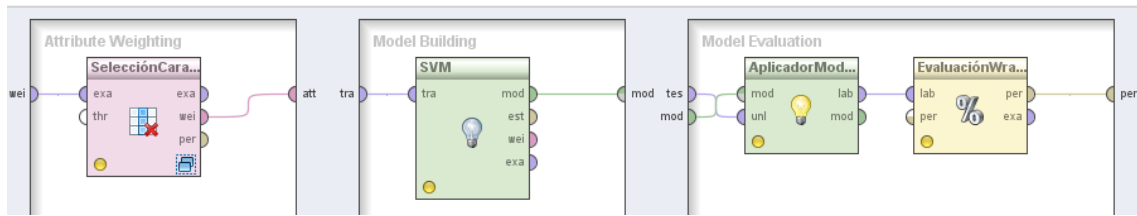
En el panel **Model Building** central, agregar el siguiente operador:

4.2 Agregar el operador **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine** y cambiar el nombre del mismo a “Aprendiz”. Conectar la entrada **tra** del panel a la entrada **tra** de este operador y la salida **mod** del mismo al conector **mod** del panel.

En el panel **Model Evaluation** de la derecha, agregar los siguientes operadores:

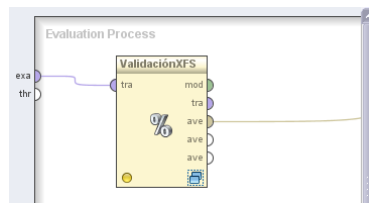
4.3 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModelo” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

4.4 **Evaluation** → **Performance Measurement** → **Classification and Regression** → **Performance (Regression)** y cambiar el nombre del mismo a “EvaluaciónWrapper”. Quitar la tilde de la opción *root mean squared error* y tildar la opción *squared error*. Conectar la salida **lab** del operador **AplicadorModelo** (Apply Model) a las entradas **lab** de este operador y la salida **per** de éste último al conector **per** del panel.



5. Hacer doble clic sobre el operador **SelecciónCaracterísticas** (Optimize Selection) del panel izquierdo. En el panel **Evaluation Process** del nivel inferior, agregar el siguiente operador:

5.1 **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “ValidaciónXFS” y el valor del parámetro *sampling type* (tipo de muestreo) a “shuffled sampling”. Conectar la entrada **exa** del panel a la entrada **tra** de este operador y la salida **ave** (averagable 1) del mismo al conector **per** del panel



6. Hacer doble clic sobre el operador **ValidaciónXFS** (X-Validation) anterior. En el panel **Training** del nivel inferior, agregar el siguiente operador:

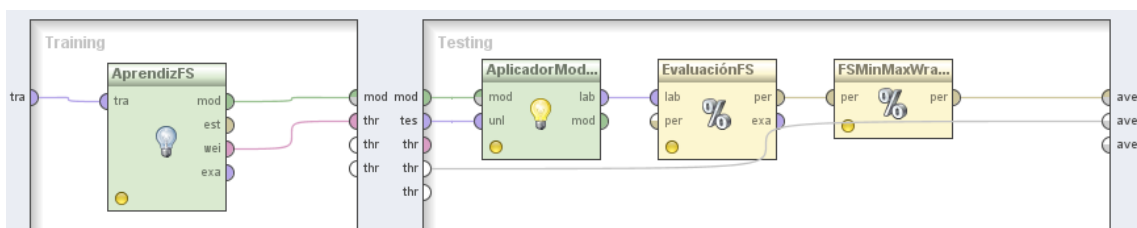
6.1 **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine** y cambiar el nombre del mismo a “AprendizFS”. Conectar la entrada **tra** del panel a la entrada **tra** de este operador y las salidas **mod** y **wei** del mismo a los conectores **mod** y **thr** del panel, respectivamente

En el panel **Testing** de la derecha, agregar los siguientes operadores:

6.2 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModeloFS” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

6.3 **Evaluation** → **Performance Measurement** → **Classification and Regression** → **Performance (Regression)** y cambiar el nombre del mismo a “EvaluaciónFS”. Quitar la tilde de la opción *root mean squared error* y tildar la opción *squared error*. Conectar la salida **lab** del operador **AplicadorModeloFS** (Apply Model) a la entradas **lab** de este operador.

6.4 **Evaluation** → **Performance Measurement** → **Performance (Min-Max)** y cambiar el nombre del mismo a “FSMinMaxWrapper” y el parámetro *minimum weights* a 0.5. Conectar la salida **per** del operador **EvaluaciónFS** (Performance (Regression)) a la entradas **per** de este operador y la salida **per** de éste último al conector **ave** (averagable 1) del panel. Conectar además el puerto **thr** de este panel al conector **ave** (averagable 2) del mismo.



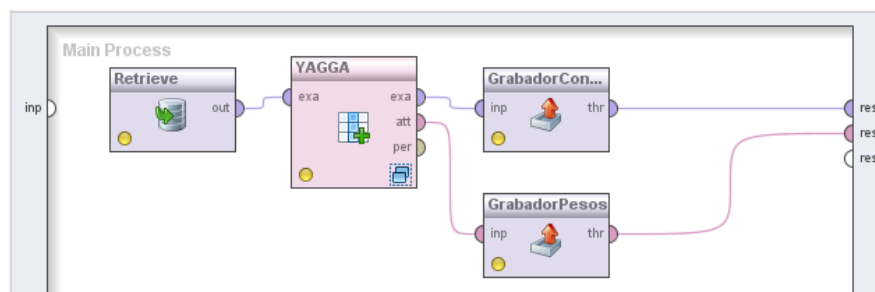
Ejemplo 17: YAGGA.

Algunas veces la selección de características sola no es suficiente. En estos casos se deben realizar otras transformaciones del espacio de características. La generación de nuevos atributos a partir de los atributos dados amplía el espacio de características. Tal vez se pueda encontrar fácilmente una hipótesis en el espacio ampliado de características.

YAGGA (Yet Another Generating Genetic Algorithm) es un wrapper híbrido de selección/generación de características. La estimación de la performance se hace con un elemento interno de validación cruzada. Por supuesto, otras formas de estimación de la performance también son posibles. La probabilidad de generación de características depende de la probabilidad para la eliminación de características. Esto asegura que la longitud media de los conjuntos de características se mantenga hasta que los conjuntos de características más cortos o más largos demuestran ser mejores.

Cuando YAGGA termina la transformación, se construyeron nuevas características. En muchos casos, este conjunto óptimo de características debería utilizarse sobre otros datos, también. Por lo tanto, el conjunto óptimo de atributos se escribe en un archivo. En el siguiente ejemplo veremos cómo se pueden utilizar estos archivos para transformar nuevos datos en la representación óptima de aprendizaje.

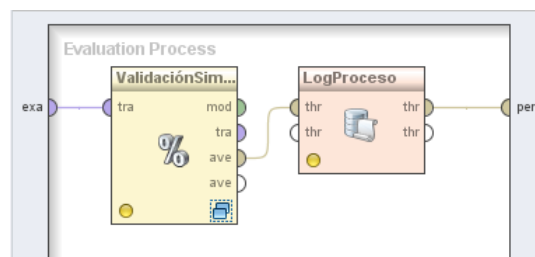
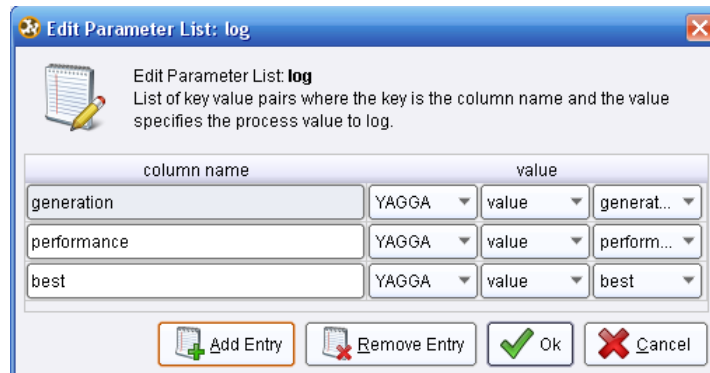
1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Polynomial con el navegador del parámetro *repository entry*.
2. Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Generation** → **Optimization** → **Optimize by Generation (YAGGA)**. Cambiar el nombre del mismo a “YAGGA” y los valores de los parámetros *population size* a 100, *maximum number of generations* a 10 y quitar la tilde de *use plus*. Conectar la salida del operador **Retrieve** a la entrada **exa** de este operador.
3. Agregar el operador **Export** → **Attributes** → **Write Constructions**. Cambiar el nombre del mismo a “GrabadorConstrucciones”. Con el navegador del parámetro *attribute constructions file* localizar la ubicación para un archivo que se va a denominar *yagga.att*. Conectar la salida **exa** del operador **YAGGA** (Optimize by Generation) a la entrada **inp** de este operador y la salida **thr** de este último al conector **res** del panel.
4. Agregar el operador **Export** → **Attributes** → **Write Weights**. Cambiar el nombre del mismo a “GrabadorPesos”. Con el navegador del parámetro *attribute weights file* localizar la ubicación para un archivo que se va a denominar *yagga.wgt*. Conectar la salida **att** del operador **YAGGA** a la entrada **inp** de este operador y la salida **thr** de este último a otro conector **res** del panel.



5. Hacer doble clic sobre el operador **YAGGA**. En el panel **Evaluation Process** del nivel inferior, agregar los siguientes operadores:

5.1 **Evaluation** → **Validation** → **Split Validation** y cambiar el nombre del mismo a “ValidaciónSimple”. Conectar la entrada **exa** del panel a la entrada **tra** (training) de este operador.

5.2 **Utility** → **Logging** → **Log**. Cambiar el nombre del mismo a “LogProceso”. Conectar la salida **ave** (averagable 1) del operador **ValidaciónSimple** (Split Validation) a la entrada **thr** (through 1) de este operador y la salida (through 1) del mismo, al conector **per** (performance) del panel. En el parámetro *log* de este operador editar la lista de parámetros para incluir los campos “generation”, “performance” y “best”:



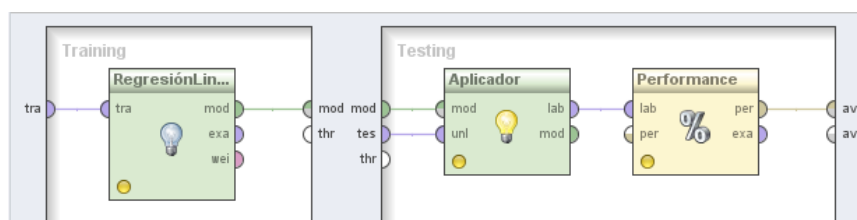
6. Hacer doble clic sobre el operador **ValidaciónSimple**. En el panel **Training** del nivel inferior, agregar el siguiente operador:

6.1 **Modeling** → **Classification and Regression** → **Function Fitting** → **Linear Regression** y cambiar el nombre del mismo a “Regresión Lineal”. Conectar la entrada **tra** y salida **mod** de este operador a los puertos **tra** y **mod** del panel, respectivamente

En el panel **Testing** de la derecha, agregar los siguientes operadores:

6.2 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “Aplicador” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

6.3 Agregar el operador **Evaluation** → **Performance Measurement** → **Classification and Regression** → **Performance (Regression)**. Conectar la salida **lab** del operador **Aplicador** (Apply Model) a la entradas **lab** de este operador y la salida **per** de éste último al conector **ave** del panel.



7. Ejecutar el proceso. Se entrega como resultado el conjunto transformado de ejemplos de entrada, la estimación de la performance, y un vector de pesos.

8. Intente agregar un operador log (registro) de proceso. Debido a que YAGGA sólo permite un operador interno, hay que agregar una cadena de un solo operador (desde el grupo “core”) a YAGGA. Hacer clic derecho sobre el operador de validación cruzada y seleccionar cortar y pegar la validación cruzada en la cadena agregada. Agregar un operador log de proceso en la cadena. Agregar los valores que desea graficar a la lista de parámetros del operador log de proceso. Consultar el Tutorial RapidMiner para más explicaciones.

- Una cadena de un solo operador para combinar varios operadores.
- Corta un operador del árbol de operadores.
- Pega un operador previamente cortado en la cadena de operadores seleccionada.

| attribute | weight |
|-----------|--------|
| a1 | 1 |
| a2 | 1 |
| a3 | 0 |
| a4 | 0 |
| a5 | 0 |
| gensym18 | 1 |
| gensym45 | 1 |
| gensym58 | 1 |
| gensym77 | 1 |

| Role | Name | Type | Statistics | Range | Missings |
|---------|----------|------|-------------------------|-------------------|----------|
| label | label | real | avg = 180.418 +/- 184.4 | [0.143 ; 926.739] | 0 |
| regular | a1 | real | avg = 5.000 +/- 2.692 | [0.081 ; 9.940] | 0 |
| regular | a2 | real | avg = 4.812 +/- 2.957 | [0.009 ; 9.986] | 0 |
| regular | gensym18 | real | avg = 23.985 +/- 21.666 | [0.029 ; 88.996] | 0 |
| regular | gensym45 | real | avg = 24.302 +/- 21.895 | [0.010 ; 85.502] | 0 |
| regular | gensym58 | real | avg = 124.576 +/- 148.5 | [0.007 ; 775.327] | 0 |
| regular | gensym77 | real | avg = 31.858 +/- 30.154 | [0.000 ; 99.715] | 0 |

Ejemplo 18: Configuración atributos resultantes de YAGGA.

En el proceso anterior se buscó un conjunto óptimo de atributos (por favor, asegurarse de ejecutar los procesos anteriores, antes de iniciar este proceso). Este conjunto óptimo de atributos se carga y se aplica a otros datos de entrada. Esto es necesario para aplicar un modelo aprendido a partir de datos con la misma representación de entrada.

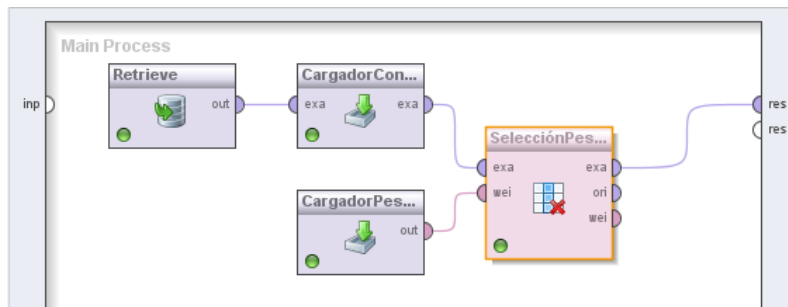
1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Polynomial con el navegador del parámetro *repository entry*.

2. Agregar el operador **Import** → **Attributes** → **Read Constructions**. Cambiar el nombre del mismo a “CargadorConstrucAtrib”. Con el navegador del parámetro *attribute constructions file* localizar el archivo *yagga.att*. Tildar la opción *keep all*. Conectar la salida del operador **Retrieve** a la entrada **exa** de este operador.

3. Agregar el operador **Import** → **Attributes** → **Read Weights**. Cambiar el nombre del mismo a “CargadorPesosAtributos”. Con el navegador del parámetro *attribute weights file* localizar el archivo *yagga.wgt*.

4. Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Selection** → **Select by Weights**. Cambiar el nombre del mismo a “SelecciónPesosAtributos”. Conectar la salida **exa** del operador **CargadorConstrucAtrib** (Read Constructions) y la salida **out** del operador **CargadorPesosAtributos** (Read Weights) a las entradas **exa** y **wei** de este operador, respectivamente. También conectar la salida **exa** de este operador al conector **res** del panel.

5. Ejecutar el proceso. Después de unos momentos, el conjunto de ejemplos de entrada utiliza la representación óptima de características que fueron encontradas en el proceso anterior.



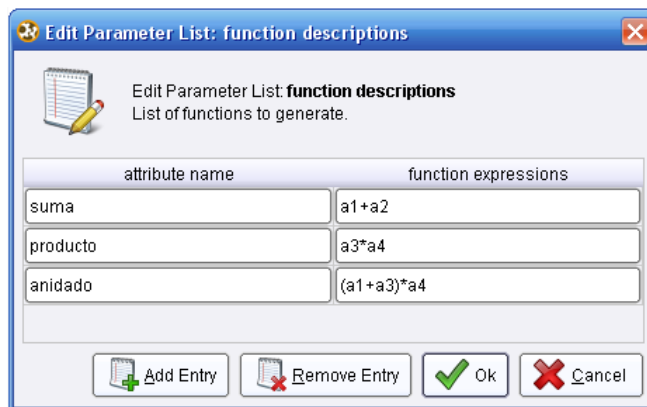
| Role | Name | Type | Statistics | Range | Missings |
|---------|----------|------|-------------------------|-------------------|----------|
| label | label | real | avg = 180.418 +/- 184.4 | [0.143 ; 926.739] | 0 |
| regular | a1 | real | avg = 5.000 +/- 2.692 | [0.081 ; 9.940] | 0 |
| regular | a2 | real | avg = 4.812 +/- 2.957 | [0.009 ; 9.986] | 0 |
| regular | gensym18 | real | avg = 23.985 +/- 21.666 | [0.029 ; 88.996] | 0 |
| regular | gensym45 | real | avg = 24.302 +/- 21.895 | [0.010 ; 85.502] | 0 |
| regular | gensym58 | real | avg = 124.576 +/- 148.5 | [0.007 ; 775.327] | 0 |
| regular | gensym77 | real | avg = 31.858 +/- 30.154 | [0.000 ; 99.715] | 0 |

Ejemplo 19: Generación de Características Definidas por el Usuario.

Este proceso carga datos numéricos desde el archivo y genera algunos atributos con el operador de generación de características. La lista de parámetros *functions* del operador de generación debe ser editada para definir las funciones que se deben generar.

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Polynomial con el navegador del parámetro *repository entry*.

2. Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Generation** → **Generate Attributes**. Cambiar el nombre del mismo a “Generación”. Conectar la salida del operador **Retrieve** a la entrada **exa** de este operador y la salida **exa** de este último al conector **res** del panel. Con el editor del parámetro *function descriptions*, agregar las siguientes funciones:



3. Ejecutar el proceso. Utilizar puntos de interrupción para comprobar el paso de generación. El parámetro *keep_all* (modo experto) define si todos los atributos deben ser utilizados para el conjunto de ejemplos resultante o sólo los atributos recientemente generados.

| Role | Name | Type | Statistics | Range | Missings |
|---------|----------|------|-------------------------|-------------------|----------|
| label | label | real | avg = 180.418 +/- 184.4 | [0.143 ; 926.739] | 0 |
| regular | a1 | real | avg = 5.000 +/- 2.692 | [0.081 ; 9.940] | 0 |
| regular | a2 | real | avg = 4.812 +/- 2.957 | [0.009 ; 9.986] | 0 |
| regular | a3 | real | avg = 5.150 +/- 2.865 | [0.001 ; 9.999] | 0 |
| regular | a4 | real | avg = 4.839 +/- 2.780 | [0.092 ; 9.950] | 0 |
| regular | a5 | real | avg = 5.035 +/- 2.976 | [0.030 ; 9.864] | 0 |
| regular | suma | real | avg = 9.812 +/- 3.980 | [0.884 ; 18.889] | 0 |
| regular | producto | real | avg = 25.501 +/- 21.889 | [0.004 ; 89.719] | 0 |
| regular | anidado | real | avg = 49.617 +/- 37.548 | [0.175 ; 185.425] | 0 |

4. Editar la lista de parámetros *functions* y agregar algunas otras funciones. Se pueden utilizar la mayoría de las funciones matemáticas conocidas.

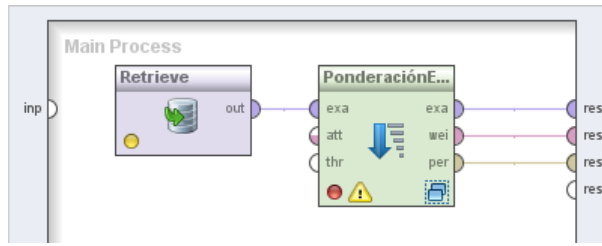
Ejemplo 20: Ponderación Evolutiva.

Este es otro proceso de ejemplo más complejo. Utiliza una cadena de validación interna (en este caso una validación simple en lugar de una validación cruzada) para estimar la performance de un aprendiz con respecto a los pesos de los atributos. Estos son adaptados con un enfoque de ponderación evolutiva.

Como se puede observar, la estructura general del proceso es muy similar a los procesos de selección y generación de características. En todos los casos se utiliza una cadena de validación interna como bloque de construcción para estimar la performance. El operador padre ("EvolutionaryWeighting" en este caso) realiza algunas operaciones sobre el conjunto de características que es evaluado por el operador hijo (validación simple).

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Weighting con el navegador del parámetro *repository entry*.

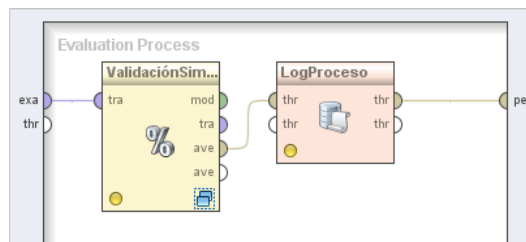
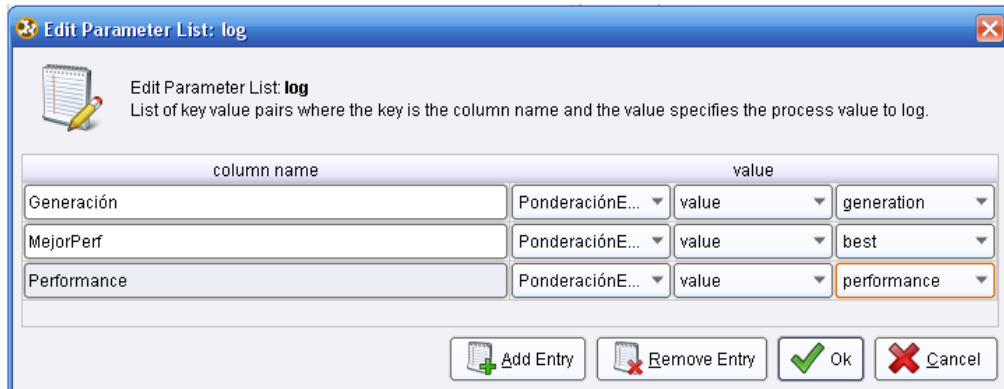
2. Agregar el operador **Modeling** → **Attribute Weighting** → **Optimization** → **Optimize Weights (Evolutionary)**. Cambiar el nombre del mismo a "Ponderación Evolutiva" y los parámetros *population size* y *maximum number of generations* a 1 y 10, respectivamente. Conectar la salida del operador **Retrieve** a la entrada **exa** de este operador, y las salidas **exa** (example set out), **wei** (weights) y **per** (performance) a conectores **res** del panel.



3. Hacer doble clic sobre el operador **PonderaciónEvolutiva** (Optimize Weights (Evolutionary)). En el panel **Evaluation Process** del nivel inferior, agregar los siguientes operadores:

3.1 **Evaluation** → **Validation** → **Split Validation**. Cambiar el nombre del mismo a “ValidaciónSimple”. Conectar la entrada **exa** del panel a la entrada **tra** (training) de este operador.

3.2 **Utility** → **Logging** → **Log**. Cambiar el nombre del mismo a “LogProceso” y conectar la salida **ave** (averagable 1) del operador **ValidaciónSimple** (Split Validation) a la entrada **thr** (through 1) de este operador y la salida (through 1) del mismo, al conector **per** (performance) del panel. En el parámetro *log* de este operador editar la lista de parámetros para incluir los campos “Generación”, “MejorPerf” y “Performance”:



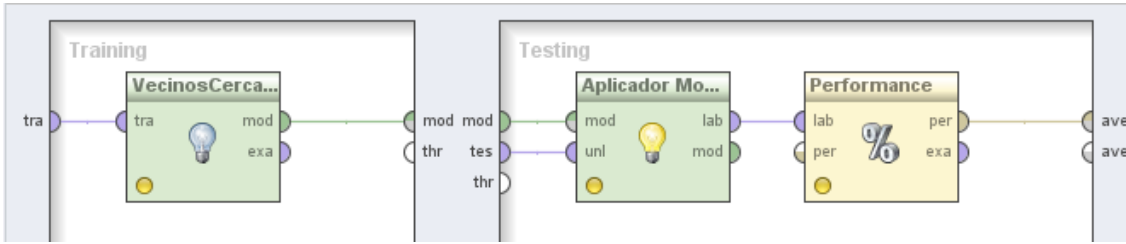
4. Hacer doble clic sobre el operador **ValidaciónSimple**. En el panel **Training** del nivel inferior, agregar el siguiente operador:


4.1 **Modeling** → **Classification and Regression** → **Lazy Modeling** → **k-NN**. Cambiar el nombre del mismo a “VecinosCercanos”. Conectar la entrada **tra** y salida **mod** de este operador a los puertos **tra** y **mod** del panel, respectivamente

En el panel **Testing** de la derecha, agregar los siguientes operadores:

4.2 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModelo” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

4.3 **Evaluation** → **Performance Measurement** → **Performance**. Conectar la salida **lab** del operador **AplicadorModelo** (Apply Model) a la entradas **lab** de este operador y la salida **per** de éste último al conector **ave** del panel.



5. Ejecutar el proceso. Cambiar a la vista “Result” y utilizar el graficador en línea. Presionar el icono “stop”  de la barra de iconos para detener el proceso. El operador actual finalizará su operación en segundo plano y puede durar algún tiempo hasta que el proceso sea detenido completamente. Aunque puede cambiar el proceso actual y reiniciarlo, se ejecutará más lento hasta que el proceso anterior sea detenido completamente.

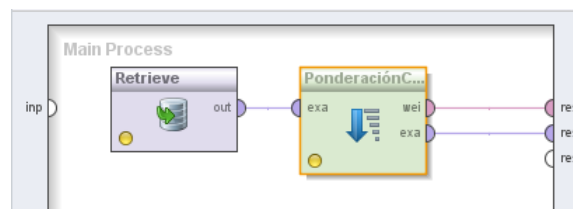
Ejemplo 21: Visualización del Conjunto de Datos y Pesos.

En este proceso se carga un conjunto de datos y se aplica uno de los esquemas de ponderación de características disponible en RapidMiner sobre este conjunto de datos. Después de que el proceso ha terminado, cambiar a la vista gráfica del conjunto de ejemplos, y observar los graficadores de alta dimensionalidad disponibles, como el gráfico paralelo, el gráfico survey, los gráficos RadViz o GridViz, matriz de histograma, matriz de cuartiles y las variantes coloreadas de estos gráficos. Notará que algunas de las columnas están marcadas con un color amarillento, por ejemplo, por un rectángulo alrededor o directamente en el gráfico. Estas marcas amarillas indican el peso de los atributos correspondientes y el color es más intenso si el peso correspondiente es mayor.

Este proceso demuestra la capacidad de RapidMiner para presentar varios resultados mediante la combinación de ellos. Por supuesto, todavía se puede tener una vista de la tabla de pesos o las diferentes vistas gráficas de los pesos de los atributos.

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Sonar con el navegador del parámetro *repository entry*.

2. Agregar el operador **Modeling** → **Attribute Weighting** → **Weight by Chi Squared Statistic**. Cambiar el nombre del mismo a “PonderaciónChiCuadrado”. Conectar la salida del operador **Retrieve** a la entrada **exa** de este operador y las salidas **wei** y **exa** de este último a conectores **res** del panel.



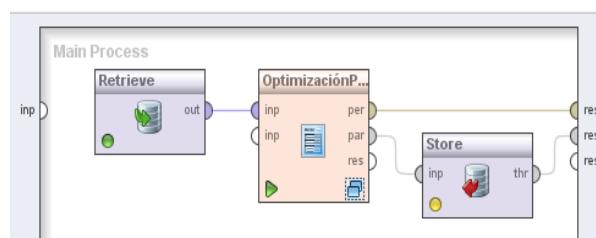
| attribute | weight |
|--------------|--------|
| attribute_1 | 0.284 |
| attribute_2 | 0.221 |
| attribute_3 | 0.083 |
| attribute_4 | 0.248 |
| attribute_5 | 0.234 |
| attribute_6 | 0.228 |
| attribute_7 | 0.099 |
| attribute_8 | 0.318 |
| attribute_9 | 0.554 |
| attribute_10 | 0.640 |
| attribute_11 | 0.966 |

Ejemplo 22: Optimización de Parámetros.

A menudo los diferentes operadores tienen muchos parámetros y no está claro qué valores de los parámetros son los mejores para la tarea de aprendizaje en cuestión. El operador de optimización de parámetros ayuda a encontrar un conjunto óptimo de parámetros para los operadores utilizados.

La validación cruzada interna estima la performance para cada conjunto de parámetros. En este proceso se afinan 2 parámetros de la SVM. El resultado puede ser graficado en 3D (utilizando gnuplot) o en modo de color.

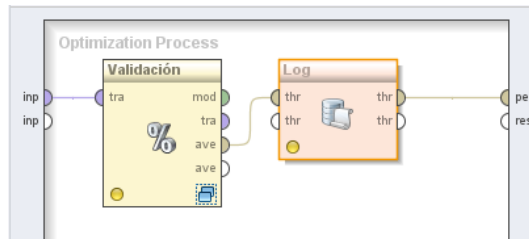
1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Polynomial con el navegador del parámetro *repository entry*.
2. **Process Control** → **Parameter** → **Optimize Parameters (Grid)**. Cambiar el nombre del mismo a “OptimizaciónParámetros”. Conectar la salida del operador **Retrieve** a la entradas **inp** (input 1) de este operador y la salida **per** de éste último al conector **res** del panel.
3. Agregar el operador **Repository Access** → **Store** a la zona de trabajo y la ruta //RapidMiner/results/Parameter-set en el parámetro *repository entry*. Cambiar el nombre del mismo a “OptimizaciónParámetros”. Conectar la salida **par** del operador **Optimize Parameters (Grid)** a la entradas **inp** (input) de este operador y la salida **thr** (through) de éste último a otro conector **res** del panel.



4. Hacer doble clic sobre el operador **OptimizaciónParámetros** (Optimize Parameters (Grid)). En el panel **Optimization Process** del nivel inferior mostrado, agregar los siguientes operadores:

4.1 **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “Validación” y el valor del parámetro *sampling type* (tipo de muestreo) a “shuffled sampling”. Conectar la entrada **inp** del panel a la entrada **tra** de este operador.

4.2 **Utility** → **Logging** → **Log**. Conectar la salida **ave** (averagable 1) del operador **Validación** (X-Validation) a la entrada **thr** (through 1) de este operador y la salida (through 1) del mismo, al conector **per** (performance) del panel.



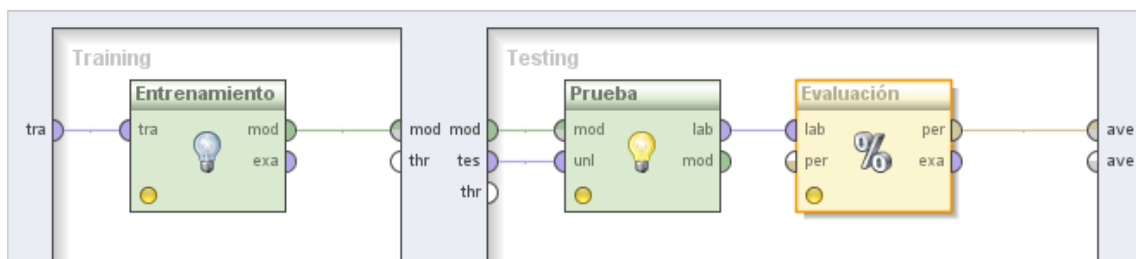
5. Hacer doble clic sobre el operador **Validación** (X-Validation). En el panel **Training** del nivel inferior, agregar el siguiente operador:

5.1 **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine (LibSVM)**. Cambiar el nombre del mismo a “Entrenamiento” y los valores de los parámetros *svm type* a “epsilon-SVR”, *kernel type* a “poly”, *degree* a 5 y *C* a 250.0. Conectar la entrada **tra** (training) y salida **mod** (model) de este operador a los puertos **tra** y **mod** del panel, respectivamente

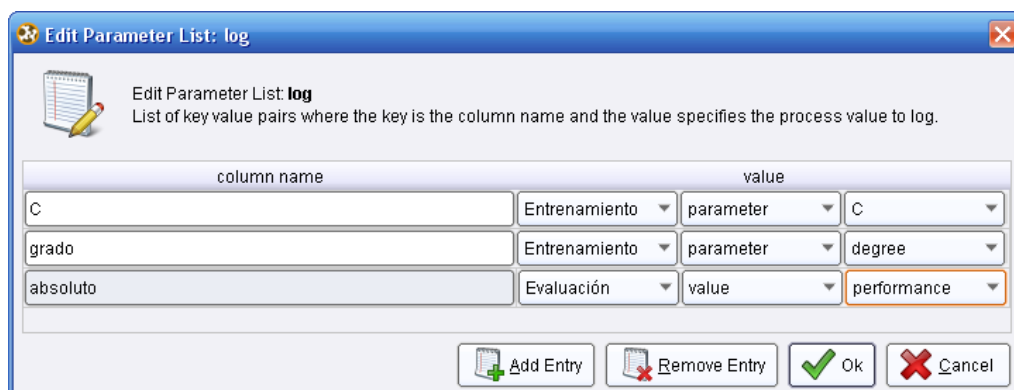
En el panel **Testing** de la derecha, agregar los siguientes operadores:

5.2 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “Prueba” y conectar las entradas **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

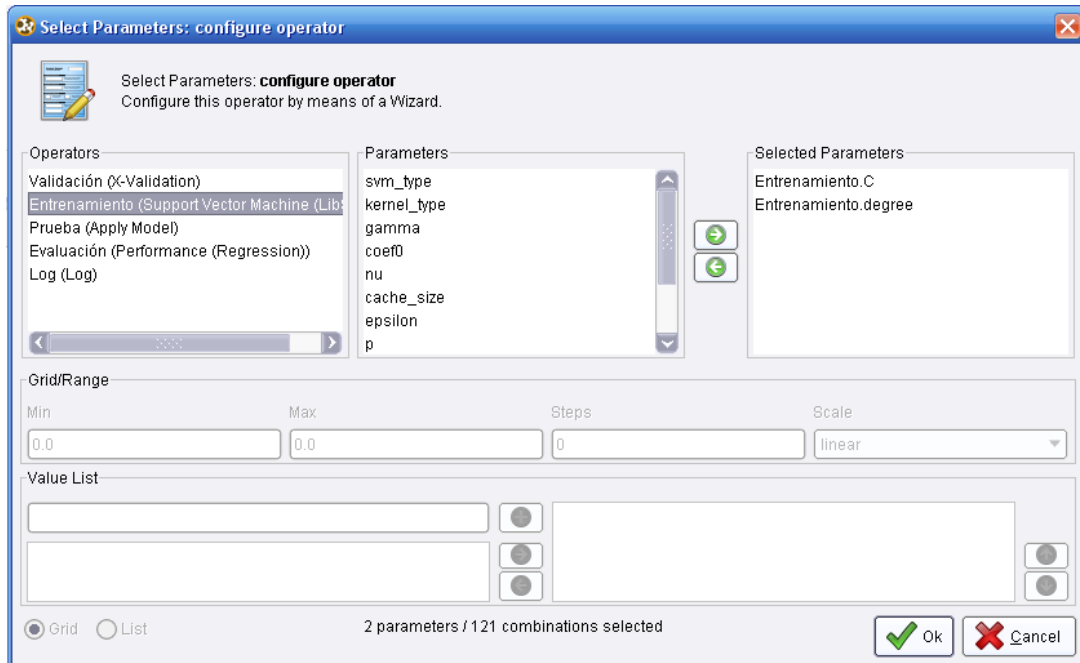
5.3 **Evaluation** → **Performance Measurement** → **Classification and Regression** → **Performance (Regression)**. Cambiar el nombre del mismo a “Evaluación” y tildar las siguientes opciones (además *root mean squared error*, tildada por defecto): *absolute error* (error absoluto) y *normalized absolute error* (error absoluto normalizado). Conectar la salida **lab** del operador **Prueba** (Apply Model) a la entrada **lab** de este operador y la salida **per** de éste último al conector **ave** del panel.



6. Subir un nivel. Seleccionar el operador **Log** y editar la lista de parámetros para incluir los campos “C”, “grado” y “absoluto” de la siguiente manera:



7. Volver al nivel superior (Proceso Principal), seleccionar el operador **OptimizaciónParámetros** y utilizar el editor para seleccionar los parámetros *C* y *degree* del operador **Entrenamiento** (Support Vector Machine (LibSVM)):



8. Ejecutar el proceso. El resultado es el mejor conjunto de parámetros y la performance lograda con ese conjunto de parámetros.

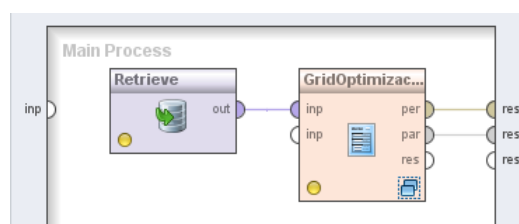
9. Editar la lista de parámetros del operador ParameterOptimization para encontrar otro conjunto de parámetros.

Ejemplo 23: Habilitador de Operadores.

Este meta-proceso muestra otra posibilidad de optimizar automáticamente el diseño del proceso. El operador "OperatorEnabler" se puede utilizar para habilitar o deshabilitar uno de sus hijos. Este se puede utilizar junto con uno de los operadores de optimización de parámetros para comprobar qué operadores se deben emplear para obtener resultados óptimos. Esto es especialmente útil para determinar qué operadores de preprocesamiento se deben usar para una combinación particular de conjunto de datos-aprendiz.

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo //Samples/data/Ripley-Set con el navegador del parámetro *repository entry*.

2. **Process Control** → **Parameter** → **Optimize Parameters (Grid)**. Cambiar el nombre del mismo a "GridOptimizaciónParámetros". Conectar la salida del operador **Retrieve** a la entradas **inp** (input 1) de este operador y las salidas **per** (performance) y **par** (parameter) de éste último conectores **res** del panel.

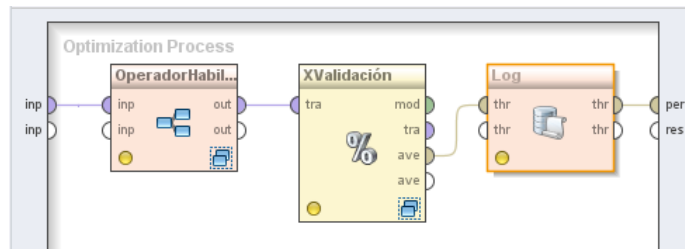
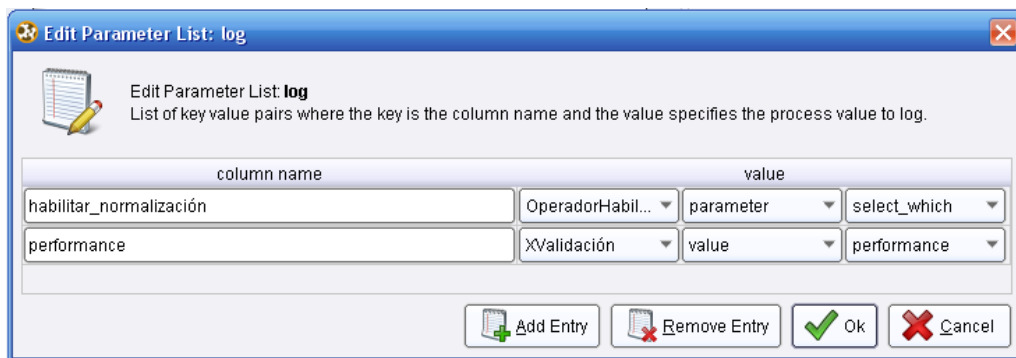


3. Hacer doble clic sobre el operador **OptimizaciónParámetros** (Optimize Parameters (Grid)). En el panel **Optimization Process** del nivel inferior mostrado, agregar los siguientes operadores:

3.1 **Process Control** → **Branch** → **Select Subprocess**. Cambiar el nombre del mismo a “OperadorHabilitador” y el valor del parámetro *select which* a 2. Conectar la entrada **inp** del panel a la entrada **inp** (input 1) de este operador.

3.2 **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “XValidación” y el valor del parámetro *number of validations* a 5. Conectar la salida **out** (output 1) del operador **OperadorHabilitador** (Select Subprocess) a la entrada **tra** de este operador.

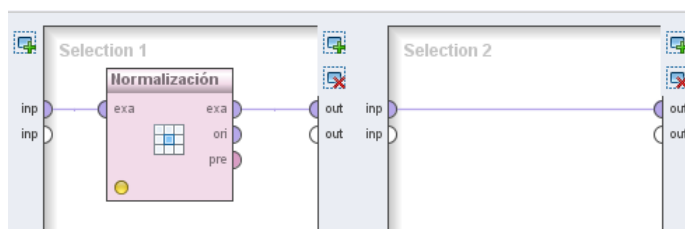
3.3 **Utility** → **Logging** → **Log**. Conectar la salida **ave** (averagable 1) del operador **XValidación** (X-Validation) a la entrada **thr** (through 1) de este operador y la salida (through 1) del mismo, al conector **per** (performance) del panel. En el parámetro *log* de este operador editar la lista de parámetros para incluir los campos “habilitar_normalización” y “performance”:



4. Hacer doble clic sobre el operador **OperadorHabilitador**. En el panel **Selection 1** del nivel inferior mostrado, agregar el siguiente operador:

4.1 **Data Transformation** → **Value Modification** → **Numerical Value Modification** → **Normalize**. Cambiar el nombre del mismo a “Normalización”. Conectar la entrada **inp** del panel a la entrada **exa** de este operador y la salida **exa** de este último al conector **out** del panel.

4.2 En el panel **Selection 2** de la derecha sólo conectar los puertos **inp** y **out** del mismo.



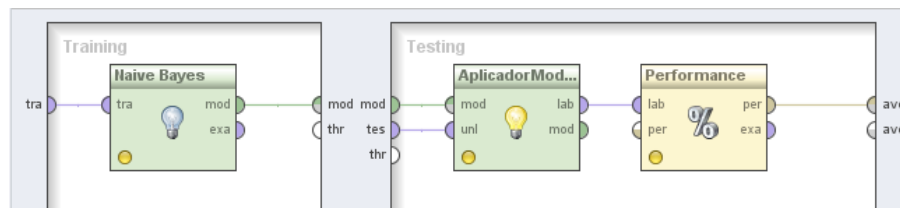
5. Subir un nivel y hacer doble clic sobre el operador **XValidación**. En el panel **Training** del nivel inferior mostrado, agregar el siguiente operador:

5.1 **Modeling** → **Classification and Regression** → **Bayesian Modeling** → **Naive Bayes**. Conectar la entrada **tra** y salida **mod** del mismo a los puertos **tra** y **mod** del panel, respectivamente.

6. En el panel **Testing** de la derecha, agregar los siguientes operadores:

6.1 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModelo” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **uni** de este operador, respectivamente.

6.2 Agregar el operador **Evaluation** → **Performance Measurement** → **Performance**. Conectar la salida **lab** del operador **AplicadorModelo** (Apply Model) a la entradas **lab** de este operador y la salida **per** de éste último al conector **ave** del panel.



7. Volver al nivel superior (Proceso Principal), seleccionar el operador **GridOptimizaciónParámetros** y utilizar el editor para seleccionar el parámetro *select which* del operador **OperadorHabilitador** (Select Subprocess):

Resultado del GridOptimizaciónParámetros (Grid):

```

Conjunto de Parámetros:
Performance:
PerformanceVector [
----accuracy: 85.20% +/- 4.12% (mikro: 85.20%)
ConfusionMatrix:
True:    0    1
0:      104   16
1:       21  109
----precision: 84.18% +/- 4.81% (mikro: 83.85%) (positive class: 1)
ConfusionMatrix:
True:    0    1
0:      104   16
1:       21  109
----recall: 87.20% +/- 8.54% (mikro: 87.20%) (positive class: 1)
ConfusionMatrix:
True:    0    1
0:      104   16
1:       21  109
----AUC (optimistic): 0.934 +/- 0.026 (mikro: 0.934) (positive class: 1)
----AUC: 0.934 +/- 0.026 (mikro: 0.934) (positive class: 1)
----AUC (pessimistic): 0.934 +/- 0.026 (mikro: 0.934) (positive class: 1)
]
OperadorHabilitador.select_which      = 1
    
```

Ejemplo 24: Umbral de Ponderación.

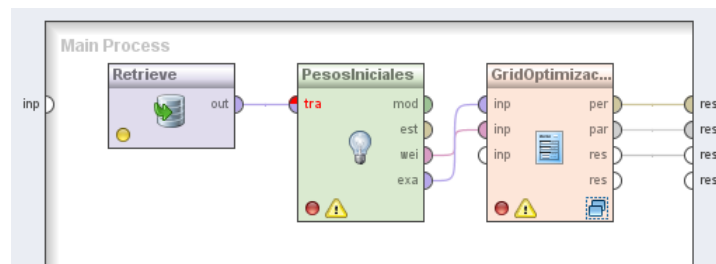
Este proceso intenta encontrar el mejor umbral de selección para los pesos proporcionados por un aprendizaje de SVM. Los pesos y el conjunto de ejemplos se pasan a un optimizador de parámetros. El parámetro

weight del operador de Selección se ha optimizado con una grid search. La performance de este umbral se evalúa con el bloque de construcción de validación cruzada. Por favor consulte los meta-procesos de ejemplos anteriores para más detalles con respecto a los operadores de optimización de parámetros.

1. Agregar el operador **Repository Access** → **Retrieve** a la zona de trabajo y localizar el archivo `//Samples/data/Weighting` con el navegador del parámetro *repository entry*.

2. Agregar el operador **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine**. Cambiar el nombre del mismo a “PesosIniciales” y quitar la tilde de la opción *scale*. Conectar la salida del operador Retrieve a la entrada **tra** de este operador.

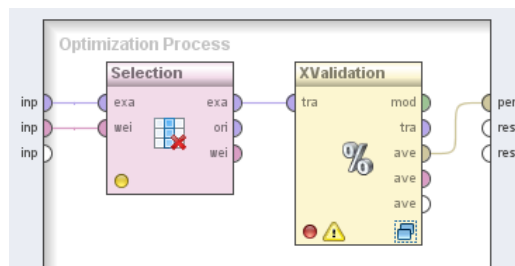
3. **Process Control** → **Parameter** → **Optimize Parameters (Grid)**. Cambiar el nombre del mismo a “GridOptimizaciónParámetros”. Conectar las salidas **exa** y **wei** del operador **PesosIniciales** (Support Vector Machine) a las entradas **inp** (input 1) e **inp** (input 2) de este operador, respectivamente, y las salidas **per** (performance), **par** (parameter) y **res** (result 1) de éste último conectores **res** del panel.



4. Hacer doble clic sobre el operador **OptimizaciónParámetros** (Optimize Parameters (Grid)). En el panel **Optimization Process** del nivel inferior mostrado, agregar los siguientes operadores:

4.1 **Data Transformation** → **Attribute Set Reduction and Transformation** → **Selection** → **Select by Weights**. Cambiar el nombre del mismo a “Selección” y el parámetro *weight* a 1.0. Conectar las entradas **inp** (input 1) e **inp** (input 2) del panel a las entradas **exa** y **wei** de este operador, respectivamente.

4.2 **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “XValidación”. Conectar la salida **exa** del operador **Selección** (Select by Weights) a la entrada **tra** de este operador y la salida **ave** (averagable 1) de este último al conector **per** del panel.



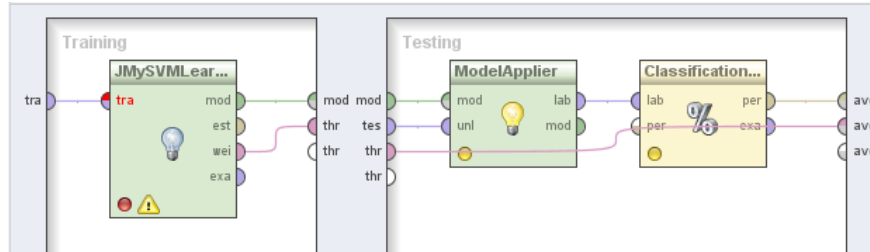
5. Hacer doble clic sobre el operador **XValidación**. En el panel **Training** del nivel inferior mostrado, agregar el siguiente operador:

5.1 **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine**. Cambiar el nombre del mismo a “JMySVM Aprendiz”. Conectar la entrada **tra** del panel a la entrada **tra** de este operador y las salidas **mod** y **wei** del mismo a los conectores **mod** y **thr** del panel, respectivamente.

6. En el panel **Testing** de la derecha, agregar los siguientes operadores:

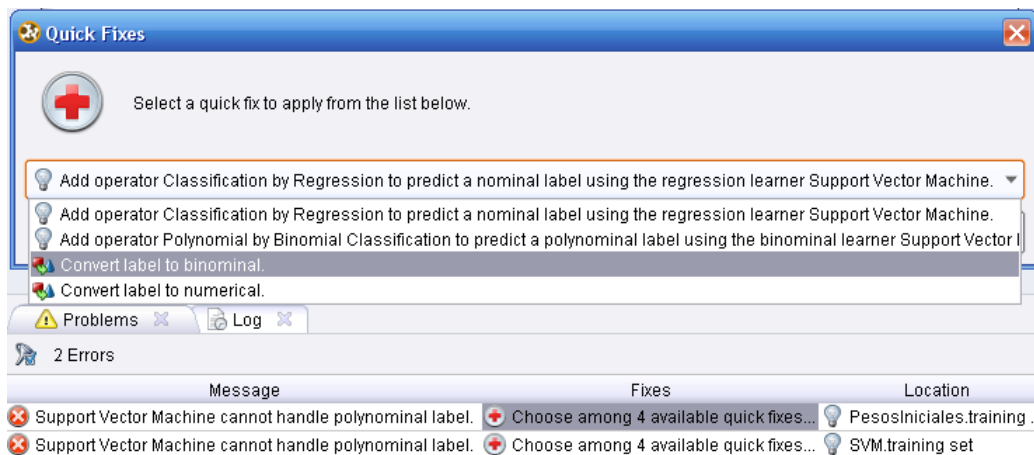
6.1 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModelo” y conectar los puertos **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

6.2 Agregar el operador **Evaluation** → **Performance Measurement** → **Classification and Regression** → **Performance (Classification)**. Cambiar el nombre del mismo a “PerformanceClasificación”, quitar la tilde de la opción “accuracy” y tildar la opción “classification error”. Conectar la salida **lab** del operador **AplicadorModelo** (Apply Model) a la entradas **lab** de este operador y la salida **per** de éste último al conector **ave** (averagable 2) del panel. También conectar la entrada **thr** (through 1) del panel a la salida **ave** (averagable 2) del mismo.



7. Volver al nivel superior (Proceso Principal), seleccionar el operador **GridOptimizaciónParámetros** y utilizar el editor (“Edit Parameter Settings...”) para seleccionar el parámetro *weight* del operador **Selección** (Select by Weights):

8. Observar los íconos de advertencia en la parte inferior izquierda de algunos operadores. Al detener un instante el puntero del ratón en la entrada **tra** del operador **PesosIniciales** del proceso principal, RapidMiner muestra que hay un error debido a que la SVM no puede manejar etiquetas polinomiales. En la pestaña Problems de la parte inferior, hacer doble clic en la primera fila, debajo de la columna “Fixes” para seleccionar una de las 4 soluciones rápidas disponibles (“Convert label to binominal.”):



Resultados:


```

Parameter set:

Performance:
PerformanceVector [
----classification_error: 2.40% +/- 2.33% (mikro: 2.40%)
ConfusionMatrix:
True:  negative      positive
negative:    235      9
positive:     3      253
]
Selección.weight    = NaN

```

| | true negative | true positive | class precision |
|----------------|---------------|---------------|-----------------|
| pred. negative | 235 | 9 | 96.31% |
| pred. positive | 3 | 253 | 98.83% |
| class recall | 98.74% | 96.56% | |

Ejemplo 25: Prueba de Significancia.

Muchos operadores de RapidMiner se pueden utilizar para estimar la performance de un aprendiz, un paso de preprocesamiento, o un espacio de características sobre uno o varios conjuntos de datos. El resultado de estos operadores de validación es un vector de performance que recoge los valores de un conjunto de criterios de performance. Para cada criterio se dan el valor medio y la desviación estándar.

La cuestión es ¿cómo se pueden comparar estos vectores de performance? Las pruebas de estadísticas de significancia como ANOVA o pruebas t por pares, se pueden utilizar para calcular la probabilidad de que los valores medios reales sean diferentes.

Suponemos que se han obtenido varios vectores de performance y se desea compararlos. En este proceso se utiliza el mismo conjunto de datos para las validaciones cruzadas (de ahí el IOMultiplier) y para estimar la performance de un esquema de aprendizaje lineal y una RBF basada en SVM.

1. Agregar el operador **Utility** → **Data Generation** → **Generate Data** a la zona de trabajo. Cambiar el nombre del mismo a “GeneradorConjEjs” y los valores de los parámetros *target function* a “one variable non linear”, *number examples* a 80, *number of attributes* a 1, *attributes lower bound* a -40.0 y *attributes upper bound* a 30.0.

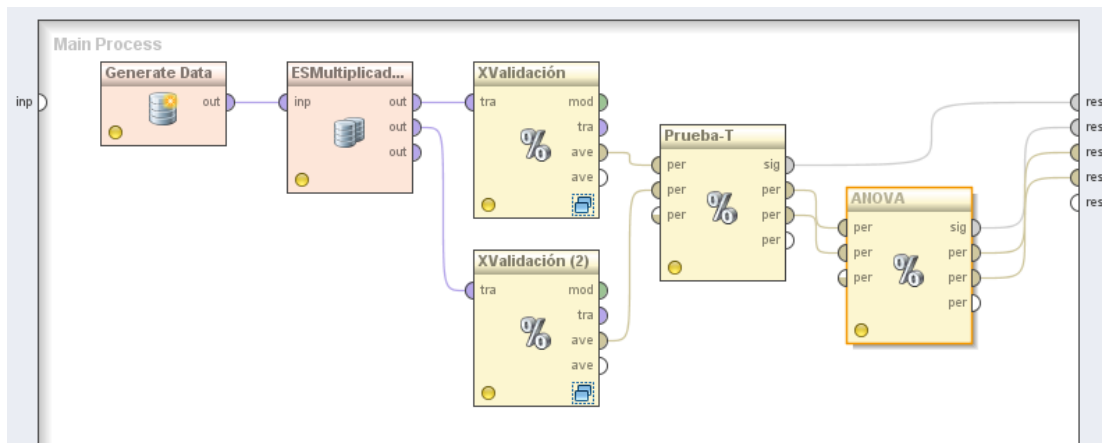
2. Agregar el operador **Process Control** → **Multiply**. Cambiar el nombre del mismo a “ESMultiplicador_1” y conectar la salida del operador **GeneradorConjEjs** (Generate Data) a la entrada de este operador.

3. Agregar un operador **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “XValidación” y el parámetro *sampling type* a “shuffled sampling”. Conectar la salida **out** (output 1) del operador **ESMultiplicador_1** (Multiply) a la entrada **tra** de este operador.

4. Agregar otro operador **Evaluation** → **Validation** → **X-Validation**. Cambiar el nombre del mismo a “XValidación (2)” y el parámetro *sampling type* a “shuffled sampling”. Conectar la salida **out** (output 2) del operador **ESMultiplicador_1** a la entrada **tra** de este operador.

5. Agregar el operador **Evaluation** → **Significance** → **T-Test**. Conectar la salida **ave** (averagable 1) del operador **XValidación (X-Validation)** a la entrada **per** (performance 1) de este operador y la salida **ave** (averagable 1) del operador **XValidación (2)** a la entrada **per** (performance 2) de este último. También conectar la salida **sig** (significance) de este operador al conector **res** del panel.

6. Agregar el operador **Evaluation** → **Significance** → **ANOVA**. Cambiar el nombre del mismo a “Prueba-T”. Conectar las salidas **per** (performance 1) y **per** (performance 2) del operador **T-Test** a las entradas **per** (performance 1) y **per** (performance 2) de este operador, y las salidas **sig** (significance), **per** (performance 1) y **per** (performance 2) del mismo a conectores **res** del panel.



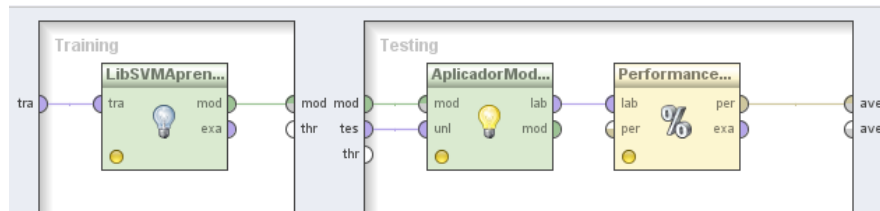
7. Hacer doble clic sobre el operador **XValidación**. En el panel **Training** del nivel inferior, agregar el siguiente operador:

7.1 **Modeling** → **Classification and Regression** → **Support Vector Modeling** → **Support Vector Machine (LibSVM)**. Cambiar el nombre del mismo a “LibSVM Aprendiz” y los valores de los parámetros *svm type* a “nu-SVR”, *kernel type* a “poly” y *C* a **10000.0**. Conectar la entrada **tra** (training) y salida **mod** (model) de este operador a los puertos **tra** y **mod** del panel, respectivamente

8. En el panel **Testing** de la derecha, agregar los siguientes operadores:

8.1 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModelo” y conectar las entradas **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

8.2 **Evaluation** → **Performance Measurement** → **Classification and Regression** → **Performance (Regression)**. Cambiar el nombre del mismo a “PerformanceRegresión”, quitar la tilde de la opción *root mean squared error* y tildar la opción *absolute error*. Conectar la salida **lab** del operador **AplicadorModelo** (Apply Model) a la entrada **lab** de este operador y la salida **per** (performance) de éste último al conector **ave** (averagable 1) del panel.



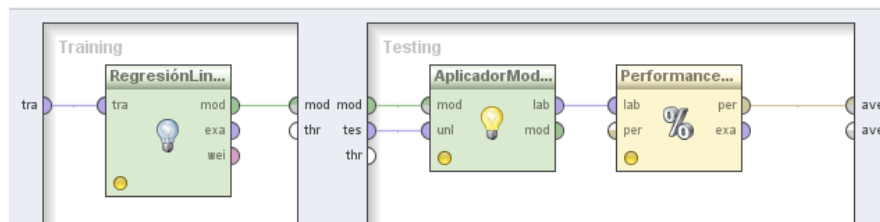
9. Subir un nivel (Proceso Principal) y hacer doble clic sobre el operador **XValidación (2)**. En el panel **Training** del nivel inferior, agregar el siguiente operador:

9.1 **Modeling** → **Classification and Regression** → **Function Fitting** → **Linear Regression**. Cambiar el nombre del mismo a “RegresiónLineal” y conectar la entrada **tra** (training) y salida **mod** (model) del mismo a los puertos **tra** y **mod** del panel, respectivamente.

10. En el panel **Testing** de la derecha, agregar los siguientes operadores:

10.1 **Modeling** → **Model Application** → **Apply Model**. Cambiar el nombre del mismo a “AplicadorModelo (2)” y conectar las entradas **mod** y **tes** del panel a las entradas **mod** y **unl** de este operador, respectivamente.

10.2 **Evaluation** → **Performance Measurement** → **Classification and Regression** → **Performance (Regression)**. Cambiar el nombre del mismo a “PerformanceRegresión (2)”, quitar la tilde de la opción *root mean squared error* y tildar la opción *absolute error*. Conectar la salida **lab** del operador **AplicadorModelo (2)** a la entrada **lab** de este operador y la salida **per** (performance) de éste último al conector **ave** (averagable 1) del panel.



11. Ejecutar el proceso y comparar los resultados: las probabilidades de una diferencia significativa son iguales, porque sólo se crearon 2 vectores de performance. En este caso, la SVM es probablemente más adecuada para el conjunto de datos en cuestión debido a que los valores medios reales probablemente son diferentes.

12. Observar que los vectores de performance como todos los demás objetos que se pueden pasar entre los operadores de RapidMiner se pueden escribir en y cargar desde un archivo.

Result Overview

PerformanceVector (PerformanceRegresión) | PerformanceVector (PerformanceRegresión (2)) | Anova Test (ANOVA) | Pairwise t-Test (Prueba-T)

ANOVA Calculator | Text View | Annotations

Anova Test

| Source | Square Sums | DF | Mean Squares | F | Prob |
|-----------|----------------|----|---------------|--------|-------|
| Between | 920165336.908 | 1 | 920165336.908 | 25.571 | 0.000 |
| Residuals | 647735048.470 | 18 | 35985280.471 | | |
| Total | 1567900385.378 | 19 | | | |

Probability for random values with the same result: 0.000
 Difference between actual mean values is probably significant, since 0.000 < alpha = 0.050!

Result Overview PerformanceVector (PerformanceRegresión (2)) PerformanceVector (PerformanceRegresión) Anova Test (ANOVA) Pairwise t-Test (Prueba-T)

T-Test Significance Text View Annotations

T-Test Significance

| | | |
|------------------------|-----------------------|------------------------|
| | 6590.265 +/- 7401.093 | 20156.144 +/- 4146.611 |
| 6590.265 +/- 7401.093 | | 0.000 |
| 20156.144 +/- 4146.611 | | |

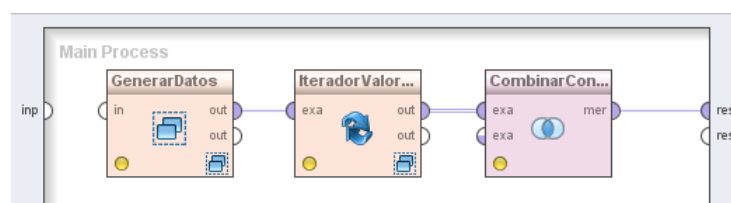
Probabilities for random values with the same result.
 Bold values are smaller than alpha=0.050 which indicates a probably significant difference between the actual mean values!

Ejemplo 26: Cálculos Basados en Grupos.

Este proceso muestra un preprocesamiento más complejo que demuestra algunas de las funcionalidades de ETL extendidas disponibles en RapidMiner mediante el uso de conceptos tales como bucles o macros.

La primera cadena de operadores sólo encapsula una secuencia de operadores que producen datos en un formato específico. Posteriormente, el *ValueIterator* itera sobre todos los valores posibles del atributo especificado, y almacena el valor actual en la macro *%{loop_value}*. Esta macro se utiliza luego dentro del *ExampleFilter* seguido por una agregación para calcular la media de otro atributo de acuerdo a los grupos definidos por el primero. Luego se utiliza otra definición de macro, *%{current_average}*, para leer la media y posteriormente se la emplea en el *AttributeConstruction*. A continuación, todos los conjuntos de datos resultantes, uno por cada grupo, se fusionarán al finalizar el bucle.

1. Agregar el operador **Utility** → **Subprocess** a la zona de trabajo. Cambiar el nombre del mismo a “GenerarDatos”.
2. Agregar el operador **Process Control** → **Loop** → **Loop Values**. Cambiar el nombre del mismo a “IteradorValores”. Conectar la salida **out** del operador **GeneradorDatos** (Subprocess) a la entrada **exa** de este operador.
3. Agregar el operador **Data Transformation** → **Set Operations** → **Append**. Cambiar el nombre del mismo a “CombinarConjEjs”. Conectar la salida **out** del operador **IteradorValores** (Loop Values) a la entrada **exa** de este operador y la salida **mer** (merged set) de este último al conector **res** del panel.



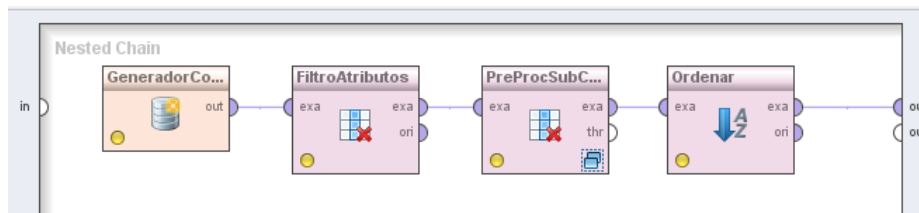
4. Hacer doble clic sobre el operador **GenerarDatos** (Subprocess). En el panel **Nested Chain** del nivel inferior, agregar los siguientes operadores:

4.1 Agregar el operador **Utility** → **Data Generation** → **Generate Data** a la zona de trabajo. Cambiar el nombre del mismo a “GeneradorConjEjs” y los valores de los parámetros *target function* a “sum”, *number examples* a 12, y *number of attributes* a 2.

4.2 Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Selection** → **Select Attributes**. Cambiar el nombre del mismo a “FiltroAtributos”. Conectar la salida **out** del operador **GeneradorConjEjs** (Generate Data) a la entrada **exa** de este operador y cambiar los valores de los parámetros *attribute filter type* a “regular_expression” y *regular expresion* a “label”, este último con ayuda del editor de expresiones regulares. Además tildar las opciones *invert selection* e *include special attributes*.

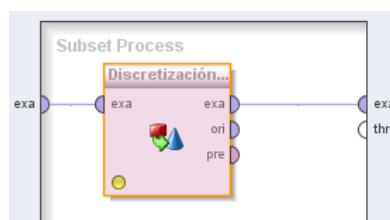
4.3 Agregar el operador **Data Transformation** → **Attribute Set Reduction and Transformation** → **Selection** → **Work on Subset**. Cambiar el nombre del mismo a “PreProcSubConjAtrib”. Conectar la salida **exa** del operador **FiltroAtributos** (Select Attributes) a la entrada **exa** de este operador y cambiar los valores de los parámetros *attribute filetr type* a “regular_expression” y *regular expresion* a “att1”, este último con ayuda del editor de expresiones regulares.

4.4 Agregar el operador **Data Transformation** → **Sorting** → **Sort**. Cambiar el nombre del mismo a “Ordenar”. Conectar la salida **exa** del operador **PreProcSubConjAtrib** (Work on Subset) a la entrada **exa** de este operador y la salida **exa** de este último al conector **out** del panel. Seleccionar “att1” de la lista de valores para el parámetro *attribute name*.



5. Hacer doble clic sobre el operador **PreProcSubConjAtrib**. En el panel **Subset Process** del nivel inferior, agregar el siguiente operador:

5.1 **Data Transformation** → **Type Conversion** → **Discretization** → **Discretize by Frequency**. Cambiar el nombre del mismo a “DiscretizaciónFrecuencias”. Conectar la entrada y salida **exa** de este operador a los conectores de entrada y salida **exa** del panel, respectivamente. Cambiar los valores de los parámetros *number of bins* a 2 y *range name type* (modo experto) a “short”.



6. Volver al proceso principal y hacer doble clic sobre el operador **IteradorValores** (Loop Values). En el panel **Iteration** del nivel inferior, agregar los siguientes operadores:

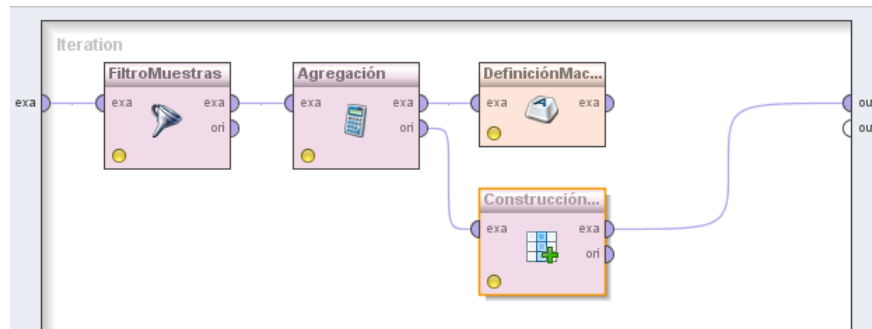
6.1 **Data Transformation** → **Filtering** → **Filter Examples**. Cambiar el nombre del mismo a “FiltroMuestras”. Conectar la entrada **exa** del panel a la entrada **exa** de este operador. Cambiar los valores de los parámetros *condition class* a “attribute_value_filter”, *parameter string* a “att1 = %{loop_value}”.

6.2 **Data Transformation** → **Aggregation** → **Aggregate**. Cambiar el nombre del mismo a “Agregación”. Conectar la entrada **exa** del operador **FiltroMuestras** (Filter Examples) a la entrada **exa** de este operador. Con la ayuda del editor de lista (Edit List) del parámetros *aggregation attributes* añadir: | att2 | average |.

6.3 **Utility** → **Macros** → **Extract Macro**. Cambiar el nombre del mismo a “DefiniciónMacroDatos”. Conectar la salida **exa** del operador **Agregación** (Aggregate) a la entrada **exa** de este operador y asignar

valores a los parámetros *macro* = “current_average”, *macro type* = “data value”, *attribute name* = “average(att2)” y *example index* = 1.

6.4 Data Transformation → Attribute Set Reduction and Transformation → Generation → Generate Attributes. Cambiar el nombre del mismo a “ConstrucciónAtributos”. Conectar la salida **ori** (original) del operador **Agregación** (Aggregate) a la entrada **exa** de este operador y la salida **exa** de este último al conector **out** del panel. Con la ayuda del editor de lista (Edit List) del parámetro *function descriptions* añadir: | *att2_avs_avg* | *abs(att2 - % $\{$ current_average $\}$)* |



7. Volver al proceso principal y seleccionar el operador **IteradorValores** (Loop Values). Seleccionar “att1” de la lista de valores para el parámetro *attribute*.

¡Felicitaciones!

Ha terminado el tutorial en línea de RapidMiner. Usted debería ser capaz de realizar muchas de las posibles definiciones de procesos. Ahora, usted conoce los bloques de construcción más importantes de las posibles definiciones de procesos de minería de datos. Por supuesto, estos bloques de construcción se pueden anidar arbitrariamente en RapidMiner siempre que sus tipos de entrada y salida sean adecuados. Para una referencia de todos los operadores, por favor consulte el Tutorial RapidMiner. También compruebe las configuraciones de los otros procesos de los ejemplos que se pueden encontrar en el directorio Sample de RapidMiner.

Hemos agregado muchos pasos de preprocesamiento conocidos y operadores de aprendizaje a RapidMiner. También se pueden manejar la mayoría de los formatos de datos. Si usted necesita adaptar RapidMiner debería leer el capítulo del Tutorial de RapidMiner que describe la creación de operadores y el mecanismo de extensión. RapidMiner se puede ampliar fácilmente. Que se divierta!

Anexo: Descripción de los Operadores utilizados en el Tutorial de RM5

1. Data Transformation → Aggregation → **Aggregate**

Este operador crea un nuevo conjunto de ejemplos a partir del conjunto de ejemplos de entrada, mostrando los resultados de las funciones de agregación arbitrarias (como SUM, COUNT, etc. conocidas del SQL). Antes de que los valores de las diferentes filas sean agregadas en una nueva fila, las filas pueden ser agrupadas por los valores de uno o varios atributos (similar a la conocida cláusula *group by* del SQL). En este caso se creará una nueva línea por cada grupo.

Tener en cuenta que se puede simular la conocida cláusula *HAVING* del SQL mediante un operador adicional *ExampleFilter* a continuación de éste.

2. Data Transformation → Attribute Set Reduction and Transformation → Generation → **Generate Attributes**

Este operador construye nuevos atributos a partir de los atributos del Conjunto de Ejemplos de entrada (*ExampleSet*) y constantes arbitrarias. Los nombres de los nuevos atributos y sus descripciones de construcción se definen en el parámetro *functions*. Los nombres de los atributos se pueden utilizar como variables en la descripción de construcciones. Cuando se evalúan las descripciones sobre cada ejemplo durante la aplicación de este operador, estas variables serán asignadas con los pesos de los atributos de los ejemplos.

Tener en cuenta que hay algunas restricciones para los nombres de los atributos para permitir que este operador trabaje correctamente:

- No están permitidos los nombres de atributos que contienen paréntesis.
- No están permitidos los nombres de atributos que contienen espacios en blanco.
- Tampoco están permitidos los nombres de atributos con nombres de función u operador.
- Las constantes estándares (ver más adelante) se pueden utilizar, los nombres de atributos con nombres como “e” o “pi” no se permiten.

Si estas condiciones no se cumplen, se deben cambiar los nombres de antemano, por ejemplo con el operador [Rename](#). Al cambiar varios atributos siguiendo un esquema determinado, puede resultar útil el operador [Rename by Replacing](#).

Expresiones soportadas

Las siguientes *operaciones* están soportadas:

- Adición: +
- Sustracción: -
- Multiplicación: *
- División: /
- Potencia: ^
- Módulo: %
- Menor que: <
- Mayor que: >
- Menor o Igual: <=
- Mayor o Igual: >=
- Igual: ==
- No Igual: !=
- Not Booleano: !
- And Booleano: &&
- Or Booleano: ||

Las siguientes *funciones logarítmicas y exponenciales* están soportadas:

- Logaritmo Natural: $\ln(x)$
- Logaritmo Base 10: $\log(x)$
- Logaritmo Dualis (Base 2): $\text{ld}(x)$
- Exponencial (e^x): $\text{exp}(x)$
- Potencia: $\text{pow}(x,y)$

Las siguientes *funciones trigonométricas* están soportadas:

- Seno: $\sin(x)$
- Coseno: $\cos(x)$
- Tangente: $\tan(x)$
- Arco Seno: $\text{asin}(x)$
- Arco Coseno: $\text{acos}(x)$
- Arco Tangente: $\text{atan}(x)$
- Arco Tangente (con 2 parámetros): $\text{atan2}(x,y)$
- Seno Hiperbólico: $\sinh(x)$
- Coseno Hiperbólico: $\cosh(x)$
- Tangente Hiperbólica: $\tanh(x)$
- Seno Hiperbólico Inverso: $\text{asinh}(x)$
- Coseno Hiperbólico Inverso: $\text{acosh}(x)$
- Tangente Hiperbólica Inversa: $\text{atanh}(x)$

Las siguientes *funciones estadísticas* están soportadas:

- Redondeo: $\text{round}(x)$
- Redondeo a p decimales: $\text{round}(x,p)$
- Piso: $\text{floor}(x)$
- Techo: $\text{ceil}(x)$
- Media: $\text{avg}(x,y,z\dots)$
- Mínimo: $\text{min}(x,y,z\dots)$
- Máximo: $\text{max}(x,y,z\dots)$

Las siguientes *funciones misceláneas* están soportadas:

- If-Then-Else: $\text{if}(\text{cond}, \text{evaluación-verdadero}, \text{evaluación-falso})$

- Valor Absoluto: `abs(x)`
- Raíz Cuadrada: `sqrt(x)`
- Signo (devuelve el signo de un número): `sgn(x)`
- Número Aleatorio (entre 0 y 1): `rand()`
- Módulo ($x \% y$): `mod(x,y)`
- Suma de k Números: `sum(x,y,z...)`
- Coeficientes Binomiales: `binom(n, i)`
- Número a Cadena: `str(x)`
- Cadena a Número: `parse(x)`
- Subcadena: `cut(x, start, len)`

Las siguientes *funciones relacionadas a procesos* están soportadas:

- Recuperar el valor de un parámetro: `param("operator", "parameter")`

Junto a los atributos y las operaciones y funciones antes mencionadas, este operador también soporta las constantes π y e si esto se especifica por el parámetro correspondiente “*use standard constants*” (por defecto: *true*). También se pueden utilizar cadenas en las fórmulas (por ejemplo, en una fórmula if-condicional), pero los valores de cadena tienen que estar encerrados entre comillas dobles (“”).

Ejemplos

```
a1+sin(a2*a3)
if (att1>5, att2*att3,-abs(att1))
```

3. Data Transformation → Attribute Set Reduction and Transformation → Generation → **Generate ID**

Este operador agrega un ID de atributo al conjunto de ejemplos dado. Cada ejemplo es etiquetado con un número entero incremental. Si el conjunto de ejemplos ya contiene un id de atributo, el atributo anterior se elimina antes de agregar el nuevo.

4. Data Transformation → Attribute Set Reduction and Transformation → Generation → Optimization → **Optimize by Generation (YAGGA)**

YAGGA es el acrónimo de Yet Another Generating Genetic Algorithm. Su enfoque para generar nuevos atributos es diferente al original. La (generación) mutación puede hacer una de las siguientes cosas con diferentes probabilidades:

- Probabilidad $p/4$: Agrega un atributo recién generado al vector de características.
- Probabilidad $p/4$: Agrega un atributo original escogido de forma aleatoria al vector de características.
- Probabilidad $p/2$: Elimina un atributo al azar del vector de características.

Así se garantiza que la longitud del vector de características pueda aumentar y disminuir. En promedio se mantendrá su longitud original, a menos que los individuos más cortos o más largos resulten tener una

mejor aptitud. Dado que este operador no contiene algoritmos para extraer características de series de valores, se limita solamente a conjuntos de ejemplos con atributos individuales. Para la extracción (automática) de características de series de valores, se debería utilizar el complemento de serie de valores para RapidMiner escrito por Ingo Mierswa. Está disponible en <http://rapid-i.com>.

5. Data Transformation → Attribute Set Reduction and Transformation → **Principal Component Análisis**

Este operador realiza un análisis de componentes principales (PCA) utilizando la matriz de covarianza. El usuario puede especificar la cantidad de varianza a cubrir en los datos originales al retener el mejor número de componentes principales. El usuario también puede especificar manualmente el número de componentes principales. El operador genera un Modelo PCA. Con el operador *ModelApplier* se pueden transformar las características.

6. Data Transformation → Attribute Set Reduction and Transformation → Selection → Optimization → **Optimize Selection**

Este operador realiza los dos algoritmos deterministas greedy (codiciosos) de selección de características: selección forward (hacia delante) y eliminación backward (hacia atrás). Sin embargo, se han añadido algunas mejoras a los algoritmos estándar, las que se describen a continuación:

Selección Forward

1. Crea una población inicial con n individuos donde n es la cantidad de atributos del conjunto de ejemplos de entrada. Cada individuo utilizará exactamente una de las características.
2. Evalúa los atributos del conjunto y selecciona sólo los k mejores.
3. Para cada uno de los k atributos del conjunto realiza: Si hay j atributos no utilizados, hace j copias del conjunto de atributos y agrega exactamente uno de los atributos anteriormente no utilizados al conjunto de atributos.
4. Mientras mejore la performance en las últimas p iteraciones se volverá al paso 2.

Eliminación Backward

1. Comienza con un conjunto de atributos con todas las características.
2. Evalúa todos atributos del conjunto y selecciona los k mejores.
3. Si hay j atributos utilizados, hace j copias del conjunto de atributos y elimina exactamente uno de los atributos utilizados anteriormente del conjunto de atributos.
4. Mientras mejore la performance en las últimas p iteraciones se volverá al paso 2.

El parámetro k puede ser especificado por el parámetro *keep_best*, el parámetro p puede ser especificado por el parámetro *generations_without_improval*. Estos parámetros tienen valores por defecto 1, lo que significa que se utilizan los algoritmos de selección estándar. Utilizando otros valores aumenta el tiempo de ejecución, pero podría ayudar a evitar extremos locales en la búsqueda del óptimo global.

Otro parámetro inusual es *maximum_number_of_generations*. Este parámetro limita el número de iteraciones a un máximo de selecciones/deselecciones de características. En combinación con

generations without improval permite varios esquemas de selección diferentes (que se describen para la selección forward, la eliminación backward trabaja de forma análoga):

- *maximum_number_of_generations* = m y *generations_without_improval* = p : Selecciona un máximo de m características. La selección se detiene si no mejora la performance medida en las últimas p generaciones.
- *maximum_number_of_generations* = -1 y *generations_without_improval* = p : Trata de seleccionar nuevas características hasta que no haya mejoras en la performance medida en las últimas p generaciones.
- *maximum_number_of_generations* = m y *generations_without_improval* = -1 : Selecciona un máximo de m características. La selección no se detiene hasta que todas las combinaciones con un máximo de m fueron probadas. Sin embargo, el resultado podría contener menos características que éstas.
- *maximum_number_of_generations* = -1 y *generations_without_improval* = -1 : Prueba todas las combinaciones de atributos (fuerza bruta, esto podría tomar un tiempo muy largo y sólo debe aplicarse a los pequeños conjuntos de atributos).

7. Data Transformation → Attribute Set Reduction and Transformation → Selection → Optimization → **Optimize Selection (Evolutionary)**

Un algoritmo genético para la selección de características (mutación = conmutar entre habilitar y deshabilitar características, cruza = intercambiar características utilizadas). La selección se realiza haciendo girar la ruleta. Los algoritmos genéticos son algoritmos de optimización/búsqueda de propósito general que son convenientes en caso de poco o ningún conocimiento del problema.

Un algoritmo genético funciona de la siguiente manera:

1. Genera una población inicial compuesta por *population_size* individuos. Cada atributo se habilita con una probabilidad *p_initialize*.
2. Para todos los individuos de la población:
 - Realizan la mutación, es decir, definen los atributos utilizados o no utilizados con probabilidad *p_mutation* y viceversa.
 - Seleccionan dos individuos de la población y realizan la cruza con probabilidad *p_crossover*. El tipo de cruza se puede seleccionar mediante *crossover_type*.
3. Realiza la selección, mapea todos los individuos a secciones de una ruleta, cuyo tamaño es proporcional a la aptitud del individuo y extrae *population_size* individuos al azar, en función de sus probabilidades.
4. Mientras mejora la aptitud, vuelve al paso 2.

Si el conjunto de ejemplos contiene atributos de series de valores con bloques de números, todo el bloque será habilitado o deshabilitado.

8. Data Transformation → Attribute Set Reduction and Transformation → Selection → **Select Attributes**

Este operador selecciona cuáles atributos de un *Conjunto de Ejemplos* deben mantenerse, y cuáles deben ser eliminados. Por lo tanto, se pueden seleccionar diferentes tipos de filtros para el parámetro *attribute filter*

type y sólo se seleccionan los atributos que satisfacen este tipo de condición. El resto será eliminado del *Conjunto de Ejemplos*. Hay un conmutador global para invertir el resultado, de modo que todos los atributos que han sido descartados inicialmente se mantendrán y viceversa. Para invertir la decisión, utilizar el parámetro *invert selection*.

Estos tipos están disponibles:

- **all:** Simplemente seleccionará todos los atributos
- **single:** Permite seleccionar un solo nombre de atributo. Este podría ser seleccionado de la lista del parámetro *attribute* si se conocen los meta datos.
- **subset:** Permite seleccionar varios atributos de una lista. No funcionará si los meta datos no están presentes. Cada atributo conocido se muestra en la lista y se podrían seleccionar.
- **regular_expression:** Permite especificar una expresión regular. Se seleccionará cada atributo cuyo nombre se corresponde con esta expresión. Las expresiones regulares son una herramienta muy potente pero necesitan una explicación detallada para los principiantes. Por favor consultar uno de los varios tutoriales disponibles en Internet para una descripción más detallada.
- **value_type:** Selecciona sólo los atributos de un determinado tipo. Tener en cuenta que los tipos son jerárquicos: Por ejemplo están los atributos denominados binominales, así como los polinominales.
- **block_type:** Similar a *value_type*, permite seleccionar los atributos en función de su tipo de bloque.
- **no_missing_values:** Seleccionará todos los atributos que no contengan un valor faltante en todos los ejemplos.
- **numeric_value_filter:** Seleccionará los atributos probando si todos los valores de sus ejemplos se corresponden con esta condición o si no son numéricos. La condición numérica se podría especificar escribiendola. Por ejemplo, la cadena de parámetro “> 6” mantendrá todos los atributos nominales y todos los atributos numéricos que tienen un valor mayor que 6 en cada ejemplo. Se puede realizar una combinación de condiciones: “> 6 && < 11” o “<= 5 || < 0”. Pero && y || no se deben mezclar.

9. Data Transformation → Attribute Set Reduction and Transformation → Selection → **Select by Weights**

Este operador selecciona todos los atributos que tienen un peso que satisface una determinada condición. Por ejemplo, sólo los atributos con un peso superior a *min_weight* deberían ser seleccionados. Este operador también es capaz de seleccionar los *k* atributos con mayor peso.

10. Data Transformation → Attribute Set Reduction and Transformation → Selection → **Work on Subset**

Este operador se puede utilizar para seleccionar un atributo (o un subconjunto de atributos) mediante la definición de una expresión regular para el nombre del atributo y aplica sus operadores internos al subconjunto resultante. Tener en cuenta que este operador también utilizará los atributos especiales, lo que lo hace necesario para todos los pasos de preprocesamiento que se deben realizar sobre los atributos especiales (y que normalmente no se realizan sobre los atributos especiales).

Este operador también es capaz de entregar los resultados adicionales del operador interno si así lo desea.

Posteriormente, los atributos originales restantes son agregados al conjunto de ejemplos resultante si el parámetro *keep_subset_only* se establece en *false* (por defecto).

Observe que este operador es muy potente y se puede utilizar para crear nuevos esquemas de preprocesamiento combinándolo con otros operadores de preprocesamiento. Sin embargo, hay dos restricciones importantes (entre algunas otras): en primer lugar, debido a que el resultado interno se combinará con el resto del conjunto de ejemplos de entrada, el número de ejemplos (puntos de datos) no se permite cambiar dentro del preprocesamiento del subconjunto. En segundo lugar, los cambios de rol de los atributos no serán entregados al exterior debido a que internamente todos los atributos especiales serán cambiados a regular para los operadores internos y los cambios de rol no se pueden entregar posteriormente.

11. Data Transformation → Attribute Set Reduction and Transformation → Transformation → **Singular Value Decomposition**

Método de reducción de dimensionalidad basada en la Descomposición Singular de Valores (SVD).

12. Data Transformation → Data Cleansing → **Replace Missing Values**

Sustituye los valores faltantes en los ejemplos. Si un valor no está presente, se reemplaza por una de las funciones “*minimum*”, “*maximum*”, “*average*”, y “*none*”, las que se aplican a los valores de los atributos que no faltan en el conjunto de ejemplos. “*none*” significa que no se sustituye el valor. La función se puede seleccionar usando la lista del parámetro *columns*. Si el nombre de un atributo aparece en esta lista como una clave, el valor se utiliza como el nombre de la función. Si el nombre del atributo no está en la lista, se utiliza la función especificada por el parámetro *default*. Para los atributos nominales se utiliza la moda para la media, es decir, el valor nominal que ocurre con mayor frecuencia en los datos. Para los atributos nominales y el tipo de reemplazo cero, se utiliza el primer valor nominal definido para este atributo. La reposición “valor” indica que se debe utilizar el parámetro definido por el usuario para la sustitución.

13. Data Transformation → Filtering → **Filter Examples**

Este operador toma un *Conjunto de Ejemplos* como entrada y devuelve un nuevo *Conjunto de Ejemplos* incluyendo sólo los Ejemplos que cumplen con una condición.

Se pueden aplicar filtros arbitrarios especificando una implementación de *Condition* y una cadena de parámetro. Los usuarios pueden implementar sus propias condiciones escribiendo una subclase de la clase anterior e implementando un constructor de 2 argumentos que toma un *Conjunto de Ejemplos* y una cadena de parámetro. Esta cadena de parámetro se especifica mediante el parámetro *parameter_string*. En lugar de utilizar una de las condiciones predefinidas, los usuarios pueden definir sus propias implementaciones con el nombre de clase completamente adecuado.

Para *attribute_value_condition* la cadena de parámetro debe tener la forma **atributo op valor**, donde **atributo** es el nombre de un atributo, **valor** es un valor que el atributo puede tomar y **op** es uno de los operadores lógicos binarios similares a los conocidos de Java, por ejemplo, mayor o igual que (\geq). Observar que se puede definir un OR lógico de varias condiciones con $\|$ y un AND lógico de dos condiciones con dos ampers and ($\&\&$) - o simplemente aplicando varios operadores *ExampleFilter* en una

fila. Tener en cuenta también que para los atributos nominales se puede definir una expresión regular para el valor de posibles comprobaciones de igualdad y desigualdad.

Para *unknown_attributes* la cadena de parámetro debe estar vacía. Este filtro elimina todos los ejemplos que contienen atributos con valores faltantes o ilegales. Para *unknown_label* la cadena de parámetro también debe estar vacía. Este filtro elimina todos los ejemplos con un valor de etiqueta desconocido.

14. Data Transformation → Name and Role Modification → **Rename**

Este operador se puede utilizar para cambiar el nombre de un atributo de un Conjunto de Ejemplos de entrada. Por favor, tener en cuenta que los nombres de atributos tienen que ser únicos.

Aunque sea renombrado, un atributo mantiene su rol. Por ejemplo, si se cambia el nombre de un atributo “etiqueta” con rol **label** a “color”, el atributo resultante “color” todavía tendrá el rol **label**. Para cambiar un rol, consultar [Set Role](#).

15. Data Transformation → Name and Role Modification → **Rename by Replacing**

Este operador sustituye partes de los nombres de atributos (como espacios en blanco, paréntesis u otros caracteres no deseados) por un reemplazo especificado. El parámetro *replace_what* se puede definir como una expresión regular (consultar el anexo del tutorial de RapidMiner para una descripción). El parámetro *replace_by* se puede definir como una cadena arbitraria. Las cadenas vacías también están permitidas. La captura de grupos de la expresión regular definida se puede acceder con *\$1*, *\$2*, *\$3* ...

16. Data Transformation → Name and Role Modification → **Set Role**

Este operador se puede utilizar para cambiar el rol de un atributo del *Conjunto de Ejemplos* de entrada. Si se desea cambiar el nombre del atributo se debe utilizar el operador [Rename](#).

El rol objetivo indica si el atributo es un atributo regular (utilizado por los operadores de aprendizaje) o un atributo especial (por ejemplo, un atributo *label* o *id*). Los siguientes tipos de atributos objetivos son posibles:

- **regular:** solo los atributos regulares se utilizan como variables de entrada para las tareas de aprendizaje.
- **id:** el atributo *id* para el conjunto de ejemplos.
- **label:** atributo objetivo para el aprendizaje.
- **prediction:** atributo pronosticado, es decir, las predicciones de un esquema de aprendizaje.
- **cluster:** indica la pertenencia a un grupo (cluster).
- **weight:** indica el peso del ejemplo.
- **batch:** indica la pertenencia a un lote (batch) de ejemplos.

Los usuarios también pueden definir tipos de atributos propios simplemente usando el nombre deseado.

¡Tener en cuenta que los roles tienen que ser únicos! Si se asigna un rol no regular por segunda vez, hará que el primer atributo sea eliminado del *Conjunto de Ejemplos*. Si se desea conservar este atributo, hay que cambiar primero su rol.

17. Data Transformation → Set Operations → **Append**

Este operador combina dos o más conjuntos dados de ejemplos agregando todos los ejemplos en una tabla de ejemplos que contiene todas las filas de datos. Tener en cuenta que la nueva tabla de ejemplos se construye en la memoria y por lo tanto este operador podría no ser aplicable a la fusión de enormes tablas de conjuntos de datos de una base de datos. En ese caso se deberían utilizar otras herramientas de preprocesamiento con tablas agregadas, unidas y fusionadas en una sola tabla que luego es utilizada por RapidMiner.

Todos los conjuntos de ejemplos de entrada deben proporcionar la misma estructura de atributos. Esto significa que todos los conjuntos de ejemplos deben tener la misma cantidad de atributos (especiales) y los mismos nombres de atributos. Si esto es cierto este operador simplemente combina todos los conjuntos de ejemplos agregando todos los ejemplos de todas las tablas en un nuevo conjunto que luego se devuelve.

18. Data Transformation → Set Operations → **Join**

Construye la unión de dos conjuntos de ejemplos utilizando los atributos id de los conjuntos, es decir, los dos conjuntos de ejemplos deben tener un atributo id donde el mismo id indica los mismos ejemplos. Si faltan ejemplos se lanzará una excepción. El conjunto de ejemplo resultante estará compuesto por la misma cantidad de ejemplos, pero el conjunto de la unión o la lista de la unión (según el ajuste del parámetro los atributos dobles serán eliminados o renombrados) de ambos conjuntos de características. En caso de eliminar los atributos duplicados los valores de los atributos deben ser los mismos para los ejemplos de ambos conjuntos de ejemplos, de lo contrario se lanzará una excepción.

Tener en cuenta que este control para atributos dobles sólo se aplicará para los atributos regulares. Los atributos especiales del segundo conjunto de ejemplos de entrada que no existen en el primer conjunto de ejemplos simplemente serán agregados. Si ya existen, simplemente son omitidos.

19. Data Transformation → Sorting → **Sort**

Este operador ordena el Conjunto de Ejemplos dado de acuerdo a un solo atributo especificado por el parámetro *attribute name*. Los ejemplos se clasifican según el orden natural de los valores de este atributo, ya sea en dirección de aumento o en disminución, dependiendo de la configuración de *sorting direction*.

20. Data Transformation → Type Conversion → Discretization → **Discretize by Frequency**

Este operador discretiza todos los atributos numéricos del conjunto de datos en atributos nominales. Esta discretización se realiza mediante intervalos de igual frecuencia, es decir, los umbrales de todos los

intervalos se seleccionan de forma que todos los intervalos contengan la misma cantidad de valores numéricos. La cantidad de intervalos se especifica mediante un parámetro, o, de forma alternativa, se calcula la raíz cuadrada de la cantidad de ejemplos sin valores faltantes (calculado para cada atributo simple). Omite todos los atributos especiales, incluyendo la etiqueta. Observe que es posible obtener intervalos con diferentes cantidades de ejemplos. Esto puede ocurrir, si los valores de los atributos no son únicos, ya que el algoritmo no puede separar entre ejemplos con el mismo valor.

21. Data Transformation → Type Conversion → Discretization → **Nominal to Binominal**

Este operador mapea los valores de todos los valores nominales a atributos binarios. Por ejemplo, si se transforma un atributo nominal con nombre “costos” y posibles valores nominales “bajo”, “moderado” y “alto”, el resultado es un conjunto de 3 atributos binominales “costos = bajo”, “costos = moderado”, y “costos = alto”. Sólo uno de los valores de cada atributo es verdadero para un ejemplo concreto, los otros valores son falsos.

22. Data Transformation → Value Modification → Numerical Value Modification → **Normalize**

Este operador realiza una normalización. Esto se puede hacer entre un valor mínimo y máximo definido por el usuario o por una transformación z, es decir, media 0 y varianza 1, o por una transformación proporcional a suma total de los atributos correspondientes.

23. Evaluation → Attributes → **Performance (Attribute Count)**

Devuelve un vector de performance que sólo cuenta la cantidad de atributos utilizados actualmente por el conjunto de ejemplos dado.

24. Evaluation → Attributes → **Performance (CFS)**

CFS evaluador de subconjuntos de atributos. Para obtener más información, consultar: Hall, M. A. (1998). Selección de subconjuntos de características basado en la correlación para Aprendizaje Automático. Tesis presentada en cumplimiento parcial de los requisitos del grado de Doctor en Filosofía de la Universidad de Waikato.

Este operador crea un filtro basado en la medida de performance para un subconjunto de características. Se evalúa el valor de un subconjunto de atributos, considerando la capacidad individual de predicción de cada característica junto con el grado de redundancia entre ellos. Se prefieren los subconjuntos de características que están altamente correlacionadas con la clase mientras tienen baja intercorrelación.

Este operador se puede aplicar sobre conjuntos de datos numéricos y nominales.

25. Evaluation → Performance Measurement → Classification and Regression → Performance (Binominal Classification)

Este operador evaluador de performance se debe utilizar para las tareas de clasificación, es decir, en los casos donde el atributo *label* tiene un tipo de valor binominal. Otras tareas de clasificación polinomial, es decir, tareas con más de dos clases pueden ser manejadas por el operador *PolynomialClassificationPerformanceEvaluator*. Este operador espera un Conjunto de Ejemplos de prueba como entrada, cuyos elementos tienen tanto las etiquetas verdaderas como las pronosticadas, y entrega como salida una lista de valores de performance que se calculan de acuerdo a una lista de criterios de performance. Si ya se dio un vector de performance de entrada, este se utiliza para mantener los valores de performance.

Todos los criterios de performance se pueden activar utilizando parámetros booleanos. Sus valores pueden ser consultados por un operador *ProcessLog* usando los mismos nombres. El criterio principal se utiliza para las comparaciones y debe ser especificado sólo para procesos donde se comparan los vectores de performance, por ejemplo, selección de características u otras configuraciones de procesos de meta optimización. Si no se selecciona ningún criterio principal, se asumirá que el criterio principal es el primer criterio del vector de performance resultante.

Los vectores de performance resultantes usualmente se comparan con un comparador de performance estándar que sólo compara los valores de aptitud del criterio principal. Se pueden especificar otras implementaciones de este comparador simple utilizando el parámetro *comparator_class*. Esto puede ser útil por ejemplo, si se desea comparar vectores de performance de acuerdo a la suma ponderada de los criterios individuales. Para implementar su propio comparador, simplemente subclase de *PerformanceComparator*. Tener en cuenta que para la optimización multi-objetivos real se suele utilizar otro esquema de selección en lugar de simplemente sustituir el comparador de performance.

26. Evaluation → Performance Measurement → Classification and Regression → Performance (Classification)

Este operador evaluador de performance se debe utilizar para tareas de clasificación, es decir, en los casos donde el atributo etiqueta tiene un tipo de valor (poli-)nominal.

Este operador espera un Conjunto de Ejemplos de prueba como entrada, que contiene un atributo con el rol *label* (etiqueta) y otro con el rol *prediction* (predicción). Consultar el operador [Set Role](#) para más detalles. Sobre la base de estos dos atributos se calcula un Vector de Performance, que contiene los valores de los criterios de performance.

Si un Vector de Performance fue alimentado en la entrada *performance*, sus valores se mantienen si no contienen nuevos criterios. De lo contrario, los valores son promediados con los valores antiguos y los nuevos.

Todos los criterios de performance se pueden activar utilizando parámetros booleanos. Sus valores pueden ser consultados por un operador *ProcessLog* usando los mismos nombres. El criterio principal se utiliza

para las comparaciones y debe ser especificado sólo para procesos donde se comparan los vectores de performance, por ejemplo, selección de atributos u otras configuraciones de procesos de meta optimización. Si no se selecciona ningún criterio principal, se asumirá que el criterio principal es el primer criterio del vector de performance resultante.

27. Evaluation → Performance Measurement → Classification and Regression → **Performance (Regression)**

Este operador evaluador de performance se debe utilizar para tareas de regresión, es decir, en los casos donde el atributo *label* (etiqueta) tiene un tipo de valor numérico. El operador espera un Conjunto de Ejemplos de prueba como entrada, cuyos elementos tienen las etiquetas verdaderas y las pronosticadas, y entrega como salida una lista de valores de performance que se calculan de acuerdo a una lista de criterios de performance. Si ya se dio un vector de performance de entrada, este se utiliza para mantener los valores de performance.

Todos los criterios de performance se pueden activar utilizando parámetros booleanos. Sus valores pueden ser consultados por un operador *ProcessLog* usando los mismos nombres. El criterio principal se utiliza para las comparaciones y debe ser especificado sólo para procesos donde se comparan los vectores de performance, por ejemplo, selección de características u otras configuraciones de procesos de meta optimización. Si no se selecciona ningún criterio principal, se asumirá que el criterio principal es el primer criterio del vector de performance resultante.

Los vectores de performance resultantes usualmente se comparan con un comparador de performance estándar que sólo compara los valores de aptitud del criterio principal. Se pueden especificar otras implementaciones de este comparador simple utilizando el parámetro *comparator_class*. Esto puede ser útil por ejemplo si se desea comparar vectores de performance de acuerdo a la suma ponderada de los criterios individuales. Para implementar su propio comparador, simplemente subclase de *PerformanceComparator*. Tener en cuenta que para la optimización multi-objetivos real se suele utilizar otro esquema de selección en lugar de simplemente sustituir el comparador de performance.

28. Evaluation → Performance Measurement → **Performance**

A diferencia de los otros métodos de evaluación de performance, como por ejemplo [Performance \(Classification\)](#), [Performance \(Binominal Classification\)](#) o [Performance \(Regression\)](#), este operador se puede utilizar para todo tipo de tareas de aprendizaje. Determinará automáticamente el tipo de tarea de aprendizaje y calculará los criterios más comunes para este tipo.

Para realizar cálculos de performance más sofisticados, debe utilizar los operadores anteriormente mencionados. Si ninguno de ellos se adapta a sus necesidades, usted podría escribir su propia medida de performance y calcularla con [Performance \(User-Based\)](#).

Este operador espera un Conjunto de Ejemplos de prueba como entrada, que contenga un atributo con el rol *label* (etiqueta) y otro con el rol *prediction* (predicción). Consultar el operador [Set Role](#) para más detalles. Sobre la base de estos dos atributos se calcula un Vector de Performance, que contiene los valores de los criterios de performance. Si un Vector de Performance fue alimentado en la entrada *performance*, sus

valores se mantienen si no contiene nuevos criterios. De lo contrario, los valores son promediados con los valores antiguos y los nuevos

Los siguientes criterios se destinan a tareas de clasificación binominal:

- Accuracy.
- Precision.
- Recall.
- AUC (optimista).
- AUC (neutral).
- AUC (pesimista).

Los siguientes criterios se destinan a tareas de clasificación polinomial:

- Accuracy.
- Kappa statistic.

Los siguientes criterios se destinan a tareas de regresión:

- Root Mean Squared Error (Raíz cuadrada del error cuadrático medio).
- Mean Squared Error (Error cuadrático medio).

29. Evaluation → Performance Measurement → **Performance (Min-Max)**

Asocia un criterio Min-Max (*MinMaxCriterion*) con cada criterio de performance de tipo Medida de Performance. Este criterio utiliza la aptitud mínima alcanzada en lugar de la aptitud media o la ponderación arbitraria de ambos. Tener en cuenta que los valores medios permanecen iguales y sólo cambian los valores de aptitud.

30. Evaluation → Performance Measurement → **Performance (User-Based)**

Este operador evaluador de performance debe ser utilizado para tareas de regresión, es decir, en los casos donde el atributo *label* tiene un tipo de valor numérico. El operador espera una Conjunto de Ejemplos de prueba como entrada, cuyos elementos tienen tanto la etiquetas verdaderas como las pronosticadas, y entrega como salida una lista de valores de performance que se calculan de acuerdo a una lista de criterios de performance. Si ya se dio un vector de performance de entrada, éste se utiliza para mantener los valores de performance.

Se pueden especificar implementaciones adicionales definidas por el usuario de los Criterios de Performance mediante la lista de parámetros *additional_performance_criteria*. Cada par clave/valor de esta lista deberá especificar un nombre de clase completamente calificado (como clave), y un parámetro de cadena (como valor) que se pasa al constructor. Por favor, asegúrese de que los archivos de clase se encuentren en la ruta de clases (este es el caso si las implementaciones son suministradas por un complemento) y que implementen un constructor de un argumento tomando un parámetro de cadena. También hay que garantizar que estas clases extiendan las *Medidas de Performance* puesto que el operador *Evaluador de Performance* sólo admitirá estos criterios. Tener en cuenta que sólo los tres primeros criterios

definidos por el usuario se pueden utilizar como valores de registro (logging) con los nombres de “user1”, ..., “user3”.

Los vectores de performance resultantes usualmente se comparan con un comparador de performance estándar que sólo compara los valores de aptitud del criterio principal. Se pueden especificar otras implementaciones de este simple comparador utilizando el parámetro *comparator_class*. Esto puede ser útil por ejemplo si se desea comparar los vectores de performance de acuerdo a la suma ponderada de los criterios individuales. Para implementar su propio comparador, simplemente subclase de *PerformanceComparator*. Tener en cuenta que para la optimización multi-objetivo real se suele utilizar otro esquema de selección en lugar de simplemente sustituir el comparador de performance.

31. Evaluation → Significance → ANOVA

Determina si la hipótesis nula (todos los valores medios reales son iguales) se cumple para los vectores de performance de entrada. Este operador utiliza un enfoque ANalysis Of VAriances (análisis de varianza) para determinar la probabilidad de que la hipótesis nula es incorrecta.

32. Evaluation → Significance → T-Test

Determina si la hipótesis nula (todos los valores medios reales son iguales) se cumple para los vectores de performance de entrada. Este operador utiliza una simple prueba t (en pares) para determinar la probabilidad de que la hipótesis nula es incorrecta. Dado que una prueba t sólo se puede aplicar a dos vectores de performance esta prueba se aplicará a todos los pares posibles. El resultado es una matriz de significancia. Sin embargo, la prueba t de a pares puede introducir un error de tipo I mayor. Se recomienda aplicar una prueba de ANOVA adicional para determinar si la hipótesis nula es totalmente incorrecta.

33. Evaluation → Validation → Split Validation

Una Cadena de Validación *RandomSplit* divide el conjunto de ejemplos en 2 conjuntos, uno de prueba y otro de entrenamiento, y evalúa el modelo. El primer operador interno debe aceptar un Conjunto de Ejemplos, mientras que el segundo debe aceptar un Conjunto de Ejemplos y la salida del primero (que en la mayoría de los casos es un modelo) y debe producir un Vector de Performance.

Este operador de validación proporciona varios valores que se pueden registrar mediante un operador [Log](#) de proceso. Todos los operadores de estimación de performance de RapidMiner facilitan el acceso a los valores medios calculados durante la estimación. Debido a que el operador no puede asegurar los nombres de los criterios entregados, el operador Log de proceso puede acceder a los valores a través de nombres genéricos de valores:

- performance: el valor del criterio principal calculado por este operador de validación.
- performance1: el valor del primer criterio del vector de performance calculado.
- performance2: el valor del segundo criterio del vector de performance calculado.
- performance3: el valor del tercer criterio del vector de performance calculado.
- para el criterio principal, también se puede acceder a la varianza y la desviación estándar en su caso.

34. Evaluation → Validation → X-Validation

X-Validation realiza un proceso de validación cruzada. La entrada *ExampleSet* S se divide en varios subconjuntos de validaciones S_i . Los subprocesos internos se aplican varias veces en validaciones usando S_i como conjunto de prueba (entrada del subproceso *Testing*) y $S \setminus S_i$ como conjunto de entrenamiento (entrada del subproceso *Training*).

El subproceso *Training* debe devolver un modelo, que suele ser entrenado con la entrada *ExampleSet*. El subproceso *Training* debe devolver un Vector de Performance. Este se suele generar aplicando el modelo y midiendo su performance. Se pueden pasar objetos adicionales desde el subproceso *Training* al *Testing* a través de los puertos.

Al igual que los otros esquemas de validación, la validación cruzada de RapidMiner puede utilizar varios tipos de muestreo para construir los subconjuntos. *Linear sampling* (muestreo lineal) simplemente divide el conjunto de ejemplos en particiones sin cambiar el orden de los ejemplos. *Shuffled sampling* (muestreo mezclado) crea subconjuntos aleatorios a partir de los datos. *Stratified sampling* (muestreo estratificado) crea subconjuntos aleatorios y asegura que la distribución de clases en los subconjuntos sea igual que en todo el conjunto de ejemplos. Para tener particiones aleatorias independientes del proceso anterior, se podría utilizar una semilla aleatoria local. Ver los parámetros para más detalles.

El operador de validación cruzada proporciona varios valores que se pueden registrar por medio de un [Log](#). Por supuesto, se puede registrar la cantidad actual de iteraciones, lo que podría ser útil para los operadores *ProcessLog* encapsulados en una validación cruzada. Además de esto, todos los operadores de estimación de performance de RapidMiner facilitan el acceso a los valores medios calculados durante la estimación. Debido a que el operador no puede asegurar los nombres de los criterios entregados, el operador *ProcessLog* puede acceder a los valores por medio de nombres genéricos de valores:

- performance: el valor del criterio principal calculado por este operador de validación.
- performance1: el valor del primer criterio del vector de performance calculado.
- performance2: el valor del segundo criterio del vector de performance calculado.
- performance3: el valor del tercer criterio del vector de performance calculado.
- para el criterio principal, también se puede acceder a la varianza y la desviación estándar en su caso.

35. Evaluation → Validation → Wrapper-X-Validation

Este operador evalúa la performance de los algoritmos de ponderación y selección de características. El primer subproceso contiene el algoritmo a evaluar. Este debe devolver un vector de pesos de atributos que luego se aplica sobre los datos de prueba. El mismo pliegue *XValidation* de los datos se utiliza para crear un nuevo modelo durante el segundo subproceso. Este modelo se evalúa en el tercer subproceso, por lo que tiene que devolver un vector de performance. Este vector de performance sirve como un indicador de performance para el algoritmo real. Esta implementación de una *MethodValidationChain* funciona de forma similar a la *XValidation*.

36. Export → Attributes → Write Constructions

Escribe todos los atributos de un conjunto de ejemplos en un archivo. Cada línea contiene la descripción de la construcción de un atributo. Este archivo se puede leer en otro proceso utilizando el *Operador de Generación de Características* o el *Cargador de Construcciones de Atributos*.

37. Export → Attributes → **Write Weights**

Escribe los pesos de todos los atributos de un Conjunto de Ejemplos en un archivo. Por lo tanto es necesario un objeto *AttributeWeights* (Pesos de los Atributos) en la entrada de este operador. Cada línea contiene el nombre de un atributo y su peso. Este archivo se puede leer en otro proceso utilizando el *Cargador de Pesos de Atributos* y el *Aplicador de Pesos de Atributos*.

38. Export → Other → **Write Parameters**

Escribe un conjunto de parámetros en un archivo. Este se puede crear mediante uno de los operadores de optimización de parámetros, por ejemplo, un operador *Optimize Parameters (Grid)*. Este se puede aplicar luego a los operadores del proceso utilizando un *ParameterSetter*.

39. Import → Attributes → **Read Constructions**

Carga un conjunto de atributos desde un archivo y construye las características deseadas. Si *keep_all* es falso, los atributos originales se eliminan antes de crear los nuevos. Esto también significa que una selección de características se lleva a cabo sólo si en el archivo se dio un subconjunto de las características originales.

40. Import → Attributes → **Read Weights**

Lee los pesos de todos los atributos de un conjunto de ejemplos desde un archivo y crea un nuevo objeto IO *AttributeWeights*. Este objeto se puede utilizar para ampliar los valores de un conjunto de ejemplos con la ayuda del operador *Aplicador de Pesos de Atributos*.

41. Import → Other → **Read Parameters**

Lee un conjunto de parámetros desde un archivo que fue escrito por un Operador de Optimización de Parámetros. Este se puede aplicar luego a los operadores del proceso utilizando un *ParameterSetter*.

42. Modeling → Association and Item Set Mining → **Create Association Rules**

Este operador genera reglas de asociación a partir de conjuntos de elementos frecuentes. En RapidMiner, el proceso de extraer conjuntos de elementos frecuentes se divide en 2 partes: en primer lugar, la generación de conjuntos de elementos frecuentes y en segundo lugar, la generación de reglas de asociación a partir de esos conjuntos.

Para generar conjuntos de elementos frecuentes, se puede utilizar, por ejemplo, el operador *FP-Growth*. El resultado será un conjunto de elementos frecuentes que se puede utilizar como entrada para este operador.

43. Modeling → Association and Item Set Mining → **FP-Growth**

Este operador calcula todos los conjuntos de elementos frecuentes de un conjunto de datos mediante la creación de una estructura de datos *FPTree* sobre la base de datos de transacciones. Esta es una copia muy comprimida de los datos que en muchos casos cabe en la memoria principal, incluso para grandes bases de datos. Todo el conjunto de elementos frecuentes se deriva de este *FPTree*. Una ventaja importante de *FP-Growth* comparado con *Apriori* es que sólo utiliza 2 escaneos de los datos y por lo tanto frecuentemente es aplicable incluso en grandes conjuntos de datos.

Observe que el conjunto de datos dado sólo puede contener atributos binominales, es decir, atributos nominales con sólo 2 valores diferentes. Sólo tiene que utilizar los operadores de preprocesamiento para transformar el conjunto de datos. Los operadores necesarios son los operadores de discretización para cambiar los tipos de valores de los atributos numéricos a nominales y el operador *Nominal2Binominal* para transformar los atributos nominales en binominales / binarios.

Los conjuntos de elementos frecuentes son extraídos de las entradas positivas de la base de datos, es decir, de los valores nominales definidos como positivos en la base de datos. Si se utiliza un archivo de descripción de atributo (.aml) para el operador *ExampleSource* este corresponde al segundo valor que se define a través de los atributos de clase o etiquetas de valor interno.

Si sus datos no especifican las entradas positivas correctamente, puede configurarlos utilizando el parámetro *positive_value*. ¡Esto sólo funciona si todos sus atributos contienen este valor!

Este operador tiene dos modos básicos de trabajo: encontrar al menos la cantidad especificada de conjuntos de elementos con mayor soporte, sin tener en cuenta el *min_support* (por defecto) o encontrar todos los conjuntos de elementos con soporte mayor que *min_support*.

44. Modeling → Attribute Weighting → Optimization → **Optimize Weights (Evolutionary)**

Este operador realiza la ponderación de características con un enfoque de estrategias evolutivas. La varianza de la mutación aditiva gaussiana pueden ser adaptada por una regla 1/5.

45. Modeling → Attribute Weighting → **Weight by Chi Squared Statistic**

Este operador calcula la relevancia de una característica obteniendo para cada atributo del Conjunto de Ejemplos de entrada el valor de la estadística chi-cuadrado con respecto al atributo de clase.

46. Modeling → Classification and Regression → Bayesian Modeling → **Naive Bayes**

Aprendiz Naive Bayes.

47. Modeling → Classification and Regression → Function Fitting → **Linear Regression**

Este operador calcula un modelo de regresión lineal. Utiliza el criterio de Akaike para la selección del modelo.

48. Modeling → Classification and Regression → Lazy Modeling → **k-NN**

Una implementación de los k vecinos más cercanos.

49. Modeling → Classification and Regression → Meta Modeling → **MetaCost**

Este operador utiliza una matriz de costos dada para obtener las predicciones de *label* según los costos de clasificación. El método usado por este operador es similar al MetaCost según lo descrito por Pedro Domingos.

50. Modeling → Classification and Regression → Meta Modeling → **Stacking**

Esta clase utiliza $n + 1$ aprendices internos y genera n modelos diferentes utilizando los n aprendices anteriores. Las predicciones de estos n modelos se toman para crear n características nuevas para el conjunto de ejemplos, que finalmente se utiliza como entrada del primer aprendiz interno.

51. Modeling → Classification and Regression → Support Vector Modeling → **Support Vector Machine**

Este aprendiz utiliza la implementación en Java de la Máquina de Vectores Soporte *mySVM* por Stefan Rüping. Este método de aprendizaje puede ser utilizado para regresión y clasificación y proporciona un algoritmo rápido y buenos resultados para muchas tareas de aprendizaje.

52. Modeling → Classification and Regression → Support Vector Modeling → **Support Vector Machine (LibSVM)**

Aplica el aprendiz [libsvm](#) por Chih-Chung Chang y Chih-Jen Lin. La SVM es un método potente para clasificación y regresión. Este operador soporta los tipos de SVM *C-SVC* y un *nu-SVC* para tareas de clasificación, así como *epsilon-SVR* y *nu-SVR* para tareas de regresión.

Además *one-class* brinda la posibilidad de aprender a partir de sólo una clase de ejemplos y luego probar si nuevos ejemplos coinciden con los conocidos. En comparación con los otros aprendices de SVM, el libsvm también soporta aprendizaje interno multiclase y la estimación de probabilidad basada en la escala de Platt para valores de confianza adecuados después de aplicar el modelo aprendido sobre un conjunto de datos de clasificación.

53. Modeling → Classification and Regression → Tree Induction → **Decision Tree**

Este operador aprende árboles de decisión, tanto de datos nominales como numéricos. Los árboles de decisión son potentes métodos de clasificación, que con frecuencia también se pueden entender fácilmente. Para clasificar un ejemplo, se recorre el árbol desde arriba hacia abajo. Cada nodo de un árbol de decisión se etiqueta con un atributo. El valor del ejemplo para este atributo determina cuál de los arcos resultantes se toma. Para atributos nominales, hay un arco que sale por cada valor posible del atributo, y para atributos numéricos los arcos salientes se etiquetan con intervalos disjuntos.

Este aprendiz de árbol de decisión funciona de forma similar a C4.5 de Quinlan o CART. En términos generales, el algoritmo de árbol de inducción trabaja de la siguiente manera. Cuando se crea un nuevo nodo en un momento determinado, se elige un atributo para maximizar el poder discriminativo de ese nodo con respecto a los ejemplos asignados al subárbol particular. Este poder discriminativo se mide por un criterio que puede ser seleccionado por el usuario (obtener información, tasa de ganancia, índice de Gini, etc.)

El algoritmo se detiene en varios casos:

- Ningún atributo alcanza un determinado umbral (*minimum_gain*).
- Se alcanza la profundidad máxima.
- Hay menos de un cierto número de ejemplos (*minimal_size_for_split*) en el subárbol actual.

Por último, se poda el árbol, es decir, se quitan las hojas que no aumentan el poder discriminativo de todo el árbol.

54. Modeling → Clustering and Segmentation → **k-Means**

Este operador representa una implementación de k-medias. Creará un atributo de cluster si todavía no está presente.

55. Modeling → Model Application → **Apply Model**

Este operador aplica un modelo a un Conjunto de Ejemplos. Los modelos suelen contener información sobre los datos con los han sido entrenados. Esta información se puede utilizar para predecir el valor de una etiqueta posiblemente desconocida, reproducir algunas transformaciones como durante el entrenamiento o realizar otros cambios. Todos los parámetros necesarios se almacenan dentro del objeto modelo.

Por favor, prestar atención al hecho de que la aplicación de los modelos necesitará los mismos atributos durante la aplicación sobre un Conjunto de Ejemplos que cuando formaron parte del Conjunto de Ejemplos en él fueron entrenados. Algunos cambios menores como la adición de atributos serían posibles, pero podrían causar graves errores de cálculo. Por favor, asegúrese de que **la cantidad de atributos, el orden, el tipo y el rol** son consistentes durante el entrenamiento y la aplicación.

Si el modelo admite vistas, es posible crear una vista en lugar de cambiar los datos subyacentes. Para indicar al operador *Apply Model* que la haga, sólo hay que habilitar el parámetro *create view*. La transformación que normalmente sería realizada directamente sobre los datos, en este caso será calculada cada vez que se requiera un valor y el resultado se devuelve sin cambiar los datos. Por favor, tener en cuenta que no todos los modelos soportan vistas.

Si se tiene que aplicar varios modelos en fila, como por ejemplo cuando se tiene que aplicar algunos modelos de preprocesamiento antes de aplicar un modelo de predicción, entonces se podrían agrupar los modelos. Esto es posible utilizando el operador [Group Models](#) de forma conveniente.

56. Modeling → Model Application → **Group Models**

Este operador agrupa todos los modelos de entrada para formar un modelo agrupado (combinado). Este modelo se puede aplicar completamente sobre nuevos datos o escrito en un archivo en otro momento. Esto podría ser útil en los casos donde los modelos de preprocesamiento y predicción deben ser aplicados en forma conjunta sobre datos nuevos y no vistos.

Este operador sustituye al agrupamiento automático de modelos conocidos de versiones anteriores de RapidMiner. El uso explícito de este operador de agrupamiento le da al usuario más control sobre el procedimiento de agrupación. Un modelo agrupado se puede desagrupar con el operador *ModelUngrouper*.

Tener en cuenta que los modelos de entrada se agregan en orden inverso, es decir, el último modelo creado, que suele ser el primero al inicio del objeto IO, se agregará a la cola como el último modelo del modelo combinado del grupo.

57. Modeling → Model Application → Thresholds → **Apply Threshold**

Este operador aplica el umbral dado a un conjunto ejemplos y mapea una predicción soft a valores crisp. Si la confianza para la segunda clase (generalmente positiva para RapidMiner) es mayor que el umbral dado, la predicción se establece a esta clase.

58. Modeling → Model Application → Thresholds → **Find Threshold**

Este operador encuentra el mejor umbral para la clasificación crisp en base a los costos definidos por el usuario.

59. Modeling → Model Application → **Ungroup Models**

Este operador desagrupa un modelo previamente agrupado (*ModelGrouper*) y entrega los modelos agrupados de la entrada.

Este operador sustituye al agrupamiento automático de modelos conocido de versiones anteriores de RapidMiner. El uso explícito de este operador de desagrupamiento le da al usuario más control sobre el procedimiento de desagrupación. Los modelos simples se pueden agrupar con el operador *ModelGrouper*.

60. Process Control → Branch → **Select Subprocess**

Este operador se puede utilizar para emplear un solo operador interno o una cadena de operadores. El parámetro *select_which* permite definir qué operador se debe utilizar. Junto con uno de los operadores de optimización de parámetros o iteración, este operador se puede utilizar para cambiar dinámicamente la configuración del proceso que pueda ser útil para probar diferentes diseños, por ejemplo, la ganancia mediante el uso de diferentes pasos de preprocesamiento o cadenas o la calidad de determinado aprendizaje.

61. Process Control → Loop → **Loop Attributes**

Este operador toma un conjunto de datos de entrada y le aplica sus operadores internos tantas veces como lo indica el número de características de los datos de entrada. Los operadores internos pueden acceder al nombre actual de la característica mediante una macro, cuyo nombre se puede especificar a través del parámetro *iteration_macro*.

El usuario puede especificar con un parámetro si este bucle debe iterar sobre todas las características o sólo sobre las características con un tipo de valor específico, es decir, sólo sobre características numéricas o sobre características nominales. También se puede especificar una expresión regular que se utiliza como filtro, es decir, los operadores internos sólo se aplican para nombres de características que coinciden con la expresión de filtro.

62. Process Control → Loop → **Loop Values**

En cada paso de iteración, este meta operador ejecuta su proceso interno para el Conjunto de Ejemplos de entrada. Esto sucederá para cada valor posible de atributo de los atributos especificados si *all* está seleccionada para el parámetro *values*. Si se selecciona *above p*, se realiza una iteración solamente para

aquellos valores que presentan una tasa de ocurrencia de por lo menos p . Esto puede ser útil, solamente si se deben considerar grandes subgrupos.

Se puede acceder al valor actual del bucle con el nombre de la macro especificada.

63. Process Control → Parameter → **Optimize Parameters (Grid)**

Este operador encuentra los valores óptimos para un conjunto de parámetros mediante una búsqueda en red (grid search). El parámetro *parameters* es una lista de pares de valores en clave donde las claves tienen la forma *nombre_operador.nombre_parámetro* y el valor es una lista de valores separados por comas (por ejemplo: 10,15,20,25) o una definición de intervalo en el formato [inicio; fin; amplitud del paso] (por ejemplo [10; 25; 5]). De forma alternativa, se puede utilizar un patrón de valores de red (grid), por ejemplo: [inicio; fin; no_steps; escale], donde *escale* identifica el tipo del patrón.

El operador devuelve un Conjunto óptimo de Parámetros que también se puede guardar en un archivo con un *Grabador de Conjunto de Parámetros*. Este conjunto de parámetros se puede leer en otro proceso utilizando un *Cargador de Conjunto de Parámetros*.

El formato de archivo del archivo de configuración de parámetros es sencillo y se puede generar fácilmente mediante aplicaciones externas. Cada línea tiene la forma:

- nombre_operador.nombre_parámetro = valor

Además del conjunto de parámetros, devuelve todos los resultados internos generados durante la ejecución que entregó la mejor performance.

Por favor consulte la sección *Procesos Avanzados / Análisis de Parámetros y Performance* para un ejemplo de aplicación. Otros esquemas de optimización de parámetros como el Operador de Optimización Evolutiva de Parámetros también pueden ser útiles si no se conocen totalmente los mejores rangos y dependencias. Otro operador que funciona de forma similar a este operador de optimización parámetros es el operador de Iteración de Parámetros. A diferencia del operador de optimización, este operador simplemente recorre todas las combinaciones de parámetros. Esto podría ser especialmente útil para los fines de graficación.

64. Process Control → **Multiply**

Este operador copia su objeto de entrada a todos los puertos de salida conectados. Mientras más puertos están conectados, más copias se generan. Tener en cuenta que los objetos se copian por referencia, por lo tanto, los datos subyacentes de los Conjuntos de Ejemplos nunca se copian (a menos que utilice un operador *Materialize Data*). Por lo tanto, la copia de objetos no es costosa. Al copiar los Conjuntos de Ejemplos sólo se copian las referencias a los atributos. Cuando se modifican o agregan atributos a un Conjunto de Ejemplos, este cambio es invisible para las otras copias. Sin embargo, si se modifican los datos en un hilo del flujo del proceso, también se modifica en las otras copias.

65. Process Control → Parameter → **Set Parameters**

Establece un conjunto de parámetros. Estos parámetros se pueden generar por un *Operador de Optimización de Parámetros* o leer mediante un *Cargador de Conjunto de Parámetros*. Este operador es útil, por ejemplo, en la siguiente situación. Si se desea encontrar los mejores parámetros para un determinado esquema de aprendizaje, por lo general también interesa el modelo generado con estos parámetros. Mientras los primeros se pueden obtener fácilmente utilizando un *Operador de Optimización de Parámetros*, el último no es posible porque el *Operador de Optimización de Parámetros* no devuelve los Objetos IO generados en su interior, sino solamente un conjunto de parámetros. Esto se debe a que el operador de optimización de parámetros no sabe nada acerca de los modelos, sino sólo sobre los vectores de performance producidos en su interior. Los vectores de performance no necesariamente requieren un modelo. Para resolver este problema, se puede utilizar un *ParameterSetter*. Por lo general, un proceso con un *ParameterSetter* contiene por lo menos dos operadores del mismo tipo, usualmente un aprendiz. Un aprendiz puede ser un operador interno del *Operador de Optimización de Parámetros* y se puede denominar “Aprendiz”, mientras que un segundo aprendiz del mismo tipo denominado “AprendizOptimo” sigue la optimización de parámetros y debe utilizar el conjunto óptimo de parámetros encontrados por la optimización.

Para hacer que el *ParameterSetter* establezca los parámetros óptimos del operador correcto, hay que especificar su nombre. Cada parámetro de la lista de parámetros *name_map* mapea el nombre de un operador que se utilizó durante la optimización (en este caso es el "Aprendiz") a un operador que ahora debe utilizar estos parámetros (en este caso es el “AprendizOptimo”).

66. Repository Access → **Retrieve**

Este operador se puede utilizar para acceder a los repositorios presentados en RapidMiner 5. Este debería sustituir a todos los accesos a archivos, porque proporciona todo el procesamiento de los metadatos, lo que facilita mucho el uso de RapidMiner. A diferencia del acceso a un archivo sin procesar, éste operador proporcionará todos los metadatos de los datos, de modo que posibilita todas las transformaciones de los metadatos.

El único parámetro *repository_entry* referencia una entrada del repositorio que se devolverá como salida de este operador. Las ubicaciones de los repositorios se resuelven en relación a la carpeta del repositorio que contiene al proceso actual. Las carpetas del repositorio están separadas mediante una barra inclinada (/), “..” hace referencia a la carpeta padre. Una barra inclinada inicial hace referencia a la carpeta raíz del repositorio que contiene al proceso actual. Una doble barra inclinada inicial se interpreta como una ruta absoluta que comienza con el nombre de un repositorio.

- “MisDatos” busca una entrada “MisDatos” en la misma carpeta que contiene el proceso actual.
- “../entrada/MisDatos” busca una entrada “MisDatos”, ubicada en la carpeta “entrada” próxima a la carpeta que contiene el proceso actual.
- “/datos/Modelo” busca una entrada “Modelo” en la carpeta “datos” de nivel superior en el repositorio que contiene al proceso actual.
- “//Samples/data/Iris”, busca el conjunto de datos “Iris” en el repositorio “Samples”.

67. Repository Access → **Store**

Este operador almacena *Objetos IO* en un lugar de un repositorio.

68. Utility → Data Generation → Add Noise

Este operador agrega atributos aleatorios y ruido blanco a los datos. Nuevos atributos aleatorios son simplemente completados con datos aleatorios que no están correlacionados en absoluto con el *label* (la etiqueta). Además, este operador puede agregar ruido al atributo *label* o a los atributos regulares. En el caso de un *label* numérico, el *label_noise* dado es el porcentaje del rango del *label* el que define la desviación estándar del ruido normalmente distribuido que se agrega al atributo *label*. Para las etiquetas nominales el parámetro *label_noise* define la probabilidad de cambiar aleatoriamente el valor de la etiqueta nominal. En el caso de agregar ruido a los atributos regulares. El parámetro *default_attribute_noise* simplemente define la desviación estándar del ruido normalmente distribuido sin utilizar el rango de valores de atributo. Usando la lista de parámetros es posible establecer diferentes niveles de ruido para diferentes atributos. Sin embargo, esto no es posible al agregar ruido a los atributos nominales.

69. Utility → Data Generation → Generate Data

Genera un conjunto aleatorio de ejemplos para propósitos de prueba. Utiliza una subclase de *TargetFunction* (Función Objetivo) para crear los ejemplos a partir de los valores de los atributos. Las funciones objetivo posibles son: *random*, *sum* (de todos los atributos), *polynomial* (de los 3 primeros atributos, grado 3), *non linear*, *sinus*, *sinus frequency* (como *sinus*, pero con frecuencias en el argumento), *random classification*, *sum classification* (como *sum*, pero positivo para *sum* positivo y negativo para *sum* negativo), *interaction classification* (positivo par *x* negativo o positivo *y* y negativo *z*), *sinus classification* (positivo para los valores positivos del seno).

70. Utility → Logging → Log

Este operador registra datos casi arbitrarios. Estos se pueden guardar en un archivo que luego se puede leer, por ejemplo por *gnuplot*. Por otra parte, los datos recogidos se pueden graficar mediante la GUI. Esto es posible incluso en tiempo de ejecución del proceso (es decir, graficación en línea).

Los parámetros de la lista *log* se interpretan de la siguiente manera: El parámetro *key* da el mismo nombre para el nombre de columna (por ejemplo, para su uso en el graficador). El parámetro *value* especifica de dónde recuperar el valor. Esto se explica mejor con un ejemplo:

- Si el valor es *operator.Evaluador.value.absolute*, el operador *ProcessLog* busca el operador con el nombre *Evaluador*. Si este operador es un *Evaluador de Performance*, tiene un valor denominado *absolute* que da el error absoluto de la última evaluación. Este valor se consulta mediante el operador *ProcessLog*.
- Si el valor es *operator.AprendizSVM.parameter.C*, el operador *ProcessLog* busca el parámetro *C* del operador denominado *AprendizSVM*.

Cada vez que se aplica el operador *ProcessLog*, todos los valores y parámetros especificados por la lista *log* se recogen y se almacenan en una fila de datos. Al finalizar el proceso, el operador escribe las filas de datos recopiladas en un archivo (si se especifica). En el modo de GUI, 2D o 3D, se generan automáticamente los

gráficos y se muestran en el visor de resultados. Por favor consultar la sección *Procesos Avanzados / Análisis de Parámetros y Performance* para un ejemplo de aplicación.

71. Utility → Macros → **Extract Macro**

Este operador (re-)define una macro para el proceso actual. Las macros serán sustituidas en las cadenas de valores de los parámetros por los valores de las macros, ver más abajo la sección Macros. A diferencia del habitual operador [Set Macro](#), este operador establece el valor de una sola macro a partir de las propiedades de un Conjunto de Ejemplos de entrada dado, por ejemplo, a partir de propiedades como la cantidad de ejemplos o atributos, o de un valor específico de dato. El nombre de la macro se debe especificar en el parámetro *macro* y la forma de recuperar el valor desde el Conjunto de Ejemplos, se debe seleccionar de *macro_type*.

Macros

Una macro definida se puede utilizar luego en todos los operadores sucesivos como valor de parámetro para los parámetros. Una macro entonces debe estar encerrada entre “%(” y “)”.

Hay varias macros predefinidas:

- **%{process_name}**: se sustituye por el nombre del proceso (sin ruta de acceso y extensión) .
- **%{process_file}**: se sustituye por el nombre de archivo del proceso (con extensión).
- **%{process_path}**: se sustituye por la ruta absoluta completa del archivo del proceso.

Además de éstas el usuario puede definir arbitrariamente otras macros que serán sustituidas por cadenas arbitrarias durante la ejecución del proceso. Tener en cuenta que también existen algunas macros cortas, por ejemplo, **%{a}** para el número de veces que se aplicó el operador actual.

Tener en cuenta además que otros operadores como muchos de los operadores de bucle como [Loop Values](#) o [Loop Attributes](#) también agregan macros específicas.

72. Utility → Macros → **Set Macro**

(Re)define macros para el proceso actual. Las macros serán sustituidas en las cadenas de valor de los parámetros por los valores de la macro definida como parámetro de este operador. A diferencia del habitual Operador de Definición de Macros, este operador sólo soporta la definición de una sola macro, por lo que se puede utilizar dentro de iteraciones de parámetro.

Hay que definir el nombre de la macro (sin encerrar entre corchetes) y el valor de la macro. La macro definida luego se puede utilizar en todos los operadores sucesivos como valor del parámetro. Una macro debe entonces estar encerrada por “MACRO_START” y “MACRO_END”.

Hay varias macros predefinidas:

- **MACRO_STARTnombre_procesoMACRO_END**: se sustituye por el nombre del proceso (sin ruta de acceso y extensión).

- `MACRO_STARTarchivo_procesoMACRO_END`: se sustituye por el nombre de archivo del proceso (con extensión).
- `MACRO_STARTruta_procesoMACRO_END`: se sustituye por la ruta absoluta completa del archivo del proceso.

Además de éstas el usuario puede definir arbitrariamente otras macros que serán sustituidas por cadenas arbitrarias durante la ejecución del proceso. Tener en cuenta que también existen algunas macros cortas, por ejemplo, `MACRO_STARTaMACRO_END` para el número de veces que se aplicó el operador actual. Por favor, consultar la sección sobre macros en el tutorial de RapidMiner. Tener en cuenta además que otros operadores como el *FeatureIterator* también agregan macros específicas.

73. Utility → Miscellaneous → **Free Memory**

Limpia los recursos de memoria no utilizados. Podría ser muy útil en combinación con el operador *MaterializeDataInMemory* después de grandes árboles de preprocesamiento utilizando gran cantidad de vistas o copias de los datos. Internamente, este operador simplemente invoca la recolección de basura del lenguaje de programación Java subyacente.

74. Utility → Miscellaneous → **Materialize Data**

Crea una copia reciente y limpia de los datos en la memoria. Podría ser muy útil en combinación con el operador *MemoryCleanUp* después de grandes árboles de preprocesamiento utilizando gran cantidad de vistas o copias de los datos.

75. Utility → **Subprocess**

Una cadena de un solo operador que puede tener una cantidad arbitraria de operadores internos. Los operadores son posteriormente aplicados y sus salidas se utilizan como entrada para el operador subsiguiente. La entrada de la cadena de operador se utiliza como entrada para el primer operador interno y la salida del último operador se utiliza como salida de la cadena de operador.

BIBLIOGRAFÍA

- RapidMiner online Tutorial. RapidMiner 5.0.
- Wiki de RapidMiner: http://rapid-i.com/wiki/index.php?title=Main_Page.