UT4-PD5

Árboles de decisión en distintas plataformas:

Rapid Miner:

Tipos de problemas: Clasificación y regresión.

Algoritmos base: C4.5

Características requeridas de atributos y label: La label tiene que ser nominal para clasificación y numérica para regresión. Los atributos pueden ser nominales y numéricos.

Parámetros que acepta, significado y opciones:

- **criterion**

Selecciona el criterio por el cuál se van a seleccionar los atributos para hacer las divisiones. Para cada criterio el valor donde dividir se optimiza según el criterio elegido. Los posibles valores son:

  - **information_gain:** Se calculan las entropías de todos los atributos y se selecciona el que tenga la menor para el split. Este método presenta un sesgo a seleccionar atributos con la mayor cantidad de valores.
  - **gain_ratio:** A Una variante de la ganancia de información que ajusta la ganancia de información para cada atributo para permitir la amplitud y uniformidad de los valores de los atributos.
  - **gini_index:** Mide la desigualdad entre las distribuciones de las características de la label. Dividir en un atributo resulta en la reducción del índice Gini promedio de los conjuntos resultantes.
  - **accuracy:** Se selecciona un atributo para la división, que maximiza la precisión del árbol.
  - **least_square:** ASe selecciona un atributo para la division que minimiza la distancia al cuadrado entre el promedio de valore en un nodo y el valor verdadero.

- **maximal_depth**

La profundidad de un árbol varía dependiendo de las características y el tamaño del dataset. Este parámetro se usa para restringir la profundidad del árbol de decisión. En caso que el valor sea -1 no hay límite para la profundidad máxima del árbol.

- **apply_pruning**

El árbol de decisión puede ser podado después de su construcción. Si se usa este parámetro algunas ramas son reemplazadas por horas según el parámetro confianza.

- **Confidence**

Específica el nivel de confianza usado para el cálculo del error pesimista durante la poda del árbol.

- **apply_prepruning**

Especifíca si se debe usar algún otro criterio de corte aparte de la profundidad máxima. Si se usa los parámetros minimal gain, minimal leaf size, minimal size of Split y número de alternativas de pre prunning son usados como criterios de core.

- **minimal_gain**

La ganancia de un nodo se calcula antes de hacer la división, el nodo se divide solo si su ganancia de información es mayor a la ganancia mínima.

- **minimal_leaf_size**

El tamaño de una hoja es el número de instancias en su sub-conjunto. El árbol es generado de modo que cada hoja tiene un número mínimo de instancias en sus hojas.

- **minimal_size_for_split**

El tamaño de un nodo es el número de ejemplos de su subconjunto. Solo aquellos nodos cuyo tamaño sea mayor o igual al minimal size of Split serán creados.

- **number_of_prepruning_alternatives**

Cuando se previene una división mediante pre prunning en un cierto nodo este parámetro ajusta el número de nodos alternativos testados para hacer la división.

Weka

Tipos de problemas: Clasificación y regresión

Algoritmos base: C4.5 (usa una implementación propia llamada 4.5)

Características requeridas de atributos y label: Los atributos pueden ser nominales o numéricos, la label debe ser nominal o numérica.

Parámetros que acepta, significado y opciones:

- MaxDepth: Determina la profundidad máxima del árbol. Por defecto es -1 que significa que el algoritmo va a setear una profundidad máxima automática.
- noPrunning: Determina si se podará o no el árbol de decisión.
- numFolds: El número de partes de datos que se usarán para podar los datos.
- minNum: Número de instancias mínimas de una hoja. El valor por defecto es una instancia por hojas.

Azure Machine Learning Studio

Tipos de problemas: Clasificación y regresión

Algoritmos base: Para boostear el árbol se utiliza este algoritmo tremiller.dvi (stanford.edu). Una breve descripción es:

1. Comienza con un conjunto vacío de aprendices débiles.
2. Para cada ejemplo de entrenamiento, agarra el output del conjunto. Esta es la suma de los outputs de todos los aprendices débiles del conjunto.
3. Calcula el gradiente de la función pérdida para cada ejemplo.

Esto depende en si es un problema de clasificación o regresión.

- En problemas de clasificación binaria se usa la pérdida logarítmica como en la regresión logística.
- En un modelo de regresión se usa la perdida cuadrada y el gradiente es la salida actual menos el objetivo

4. Usa los ejemplos para entrenar un aprediz débil usando el gradiente definido como objetivo en el paso anterior.
5. Añade ese aprendiz débil al conjunto con una fuerza indicada por el ratio de aprendizaje y si se desea se puede volver al paso dos.

En esta implementación, los aprendices débiles son los mínimos cuadrados árboles de regresión basados en los gradientes calculados en el paso 3. Los árboles tienen restricciones:

- Son entrenados hasta una cantidad máxima de hojas.
- Cada hoja tiene un número mínimo de ejemplos para evitar el sobre ajuste.
- Cada nodo de decisión es un atributo que se compara con un umbral. Si el atributo es menor o igual que el umbral baja por un camino y si es mayor que el umbral baja por el otro camino.
- Cada nodo hoja es un valor constante.

6. El algoritmo de construcción del árbol selecciona de forma ávida el atributo para el cual una división minimiza el error cuadrado según el gradiente calculado en el paso 3. La selección de la división está sujeta al mínimo número de instancias por hoja.

El algoritmo divide hasta que llega al número máximo de hojas o hasta que no hay una división válida.

Para problemas de regresión:

El conjunto de árboles se produce calculando en cada paso un árbol de regresión que aproxima el gradiente de la función costo y lo añade al árbol anterior con coeficientes que minimizan la pérdida de un árbol nuevo. La salida del conjunto es la suma de los tres outputs:

- Para problemas de clasificación binaria la salida es convertida a probabilidad usando una forma de calibración.
- Para problemas de regresión, la salida es el valor predicho de la función.

Características requeridas de atributos y label: Los atributos pueden ser nominales o numéricos, los atributos numéricos no continuos deben ser convertidos a valores nominales. La label debe ser nominal.

Parámetros que aceptan, significado y opciones:

Maximum number of leaves per tree: Especifica el número máximo de hojas por árbol

Minimum number of samples per leaf node: Especifica el número de casos requeridos para formar un nodo hoja.

Learning rate: Especifica el ratio de aprendizaje inicial

Total number of tres constructed: Especifica el número total de árboles construidos durante el entrenamiento.

Random number seed: Provee una semilla para el experimento que permite realizarlo varias veces en las mismas condiciones.

Allow unknown categorical levels: Si es verdadero crea un nivel adicional para cada atributo categórico. Los niveles en el datset de prueba que no están disponibles en el de entrenamiento son mapeados a este nivel adicional.

KNIME

Tipos de problemas: Clasificación

Algoritmos base: C4.5

Características requeridas de atributos y label: Los atributos pueden ser numéricos o nominales, la label debe ser nominal.

Parámetros que acepta, significado y opciones:

Class column

Selecciona la label

Quality measure

Selecciona entre indice Gini y Gain Ratio para calcular las divisiones.

Pruning method

Selecciona el método de poda entre minimal description length o ninguno.

Reduced Error Pruning

Si está elegido se usa un método simple para podar el árbol en post procesamiento. Empieza por las hojas donde cada nodo se reemplaza con su clase más frecuente, pero solo si no disminuye la precisión de las predicciones.

Min number records per node

Valor mínimo de instancias requeridas para cada nodo. Si el número de nodos es menor o igual el árbol no crece más.

Number records to store for view

Selecciona el número de instancias guardadas en el árbol para su vista

Average split point

Si está habilitado el valor de división para valores numéricos se determina de acuerdo con el promedio de los dos atributos que separan las dos divisiones. Si no está habilitado el valor de división es el mayor valor de la partición más pequeña.

Number threads

Setea el número de hilos a usar, el default es el número de núcleos asignados a knime, si hay solo uno el trabajo es secuencial.

Skip nominal columns without domain information

Si está habilitado las columnas nominales que no contienen información de valor del dominio son saltadas. Se usa en columnas nominales con muchos valores.

Force root split column

Si está habilitado la primera división es calculada en la columna elegida sin evaluar a las otras posibilidades.

Binary nominal splits

Si está habilitado los atributos nominales se dividen de forma binaria. Los splits binarios son más difíciles de calcular, pero aumentan la precisión del modelo. Los valores nominales se dividen en dos conjuntos. Si no está habilitado para cada valor nominal se crea un hijo.

Max #nominal

Los subconjuntos para los splits binarios nominales son dificiles de calcular. Para encontrar los mejores subconjuntos para n valores nominales se tienen que hacer 2^n cálculos. Para evitar este costo tan alto se puede setear este valor, por encima de este valor se aplica una heurística para calcular el mejor valor nominal para la segunda partición, luego el segundo mejor valor y así sucesivamente hasta que no hay más progreso.

Filter invalid attribute values in child nodes

Esta opción hace que se post-procese el árbol filtrando checkeos repetidos en nodos hijos.

No true child strategy

Si el puntaje alcanza un nodo cuyo valor de atributos es desconocido se puede usar una de las siguientes estrategias: ReturnNullPrediction: predice un missing value o ReturnLastPrediction: retorna la clase mayoritaria del último nodo.

Missing value strategy

Si hay valores faltantes en los datos a ser predichos se puede seleccionar una estrategia para manejarlo: LastPrediction: Usa la última predicción conocida o defaultChild: usa un hijo default y continua en su camino o NONE donde usa noTrueChildStrategy

Python Sci kit learn

Tipos de problemas: Clasificación y regresión

Algoritmos base: CART es la implementación que se usa.

Características requeridas de atributos y label: La label debe ser nominal binaria, multiclase o numérica. El algoritmo no soporta variables categóricas.

Parámetros que acepta, significado y opciones:

criterion*{"gini", "entropy"}, default="gini"*

La función para medir la calidad de una división. Las opciones son "gini" para la impureza de Gini o "entropy" para ganacia de información.

splitter*{"best", "random"}, default="best"*

La estrategia usada para elegir la división en cada nodo. Soporta "best" para elegir la mejor división o "random" para elegir una división aleatoria.

max_depth*int, default=None*

La profundidad máxima del árbol. Si no se selecciona nada los nodos se expanden hasta que todas las hojas sean puras o hasta que todas las hojas contengan menos instancias que el min_samples_split.

min_samples_split *int or float, default=2*

El mínimo número de instancias requeridas para dividir un nodo interno.

Si es un int entonces se considera min_samples_split como el número mínimo. Si es un float entonces ese valor es una fracción y ceil(min_samples_split * n_samples) es el mínimum number of samples para cada división.

min_samples_leaf*int or float, default=1*

El valor minimo de instancias requeridas para ser un nodo hoja. Un punto de division en cualquier profundidad va ser considerado como tal si sus hijos tienen por lo menos min_samples_leaf instancias.

min_weight_fraction_leaf*float, default=0.0*

La fracción ponderada minima de la suma total de pesos que debe tener un nodo para ser una hoja, cuando sample_weight no se setea todos los nodos pesan los mismo.

max_features *int, float or {"auto", "sqrt", "log2"}, default=None*

El número de atributos a considerar en la búsqueda por la mejor división. So es un int se considera max_features en cada división, si es un float se considera int(max_features * n_features) en cada división. En caso de seleccionar la opción "auto" o "sqrt" entonces max_features = sqrt(n_features), con "log2" entonces max_features = $\log_2$(n_features), en caso de ser none entonces max_features = n_features.

random_state*int, RandomState instance or None, default=None*

Controla el estimador de aleatoriedad. Los atributos siempre se permutan de forma aleatoria en cada división, incluso si el splitter está en "best". Cuando max_features < n_features el algoritmo seleccionará max_features (cantidad) atributos de forma aleatoria en cada división antes de encontrar la mejor división. La mejor división puede variar a traves de distintas iteraciones incluso si max_features = n_features, en ese caso si el criterio de mejora es el mismo para varias divisiones se tiene que elegir una de forma aleatoria, para obtener comportamiento determinístico random state tiene que ser un integer.

max_leaf_nodes *int, default=None*

Los mejores nodos se definen como la reducción de impureza relativa. Si está en none entonces hay un número ilimitado de nodos hoja.

min_impurity_decrease *float, default=0.0*

Un nodo va a ser una división si reduce la impureza más o igual que este valor.

class_weight*dict, list of dict or "balanced", default=None*

Pesos asociados a las clases en la forma {etiqueta_clase : peso}. Si no se usa ninguno entonces todas las clases tienen peso uno. En los casos en que se tenga multi-output se puede pasar una lista de valores en el mismo orden que las columnas de y, en estos casos se debe definir el peso para todas las clases de cada columna. Ejemplo: para multi-lables de 4 clases debe ser [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}].

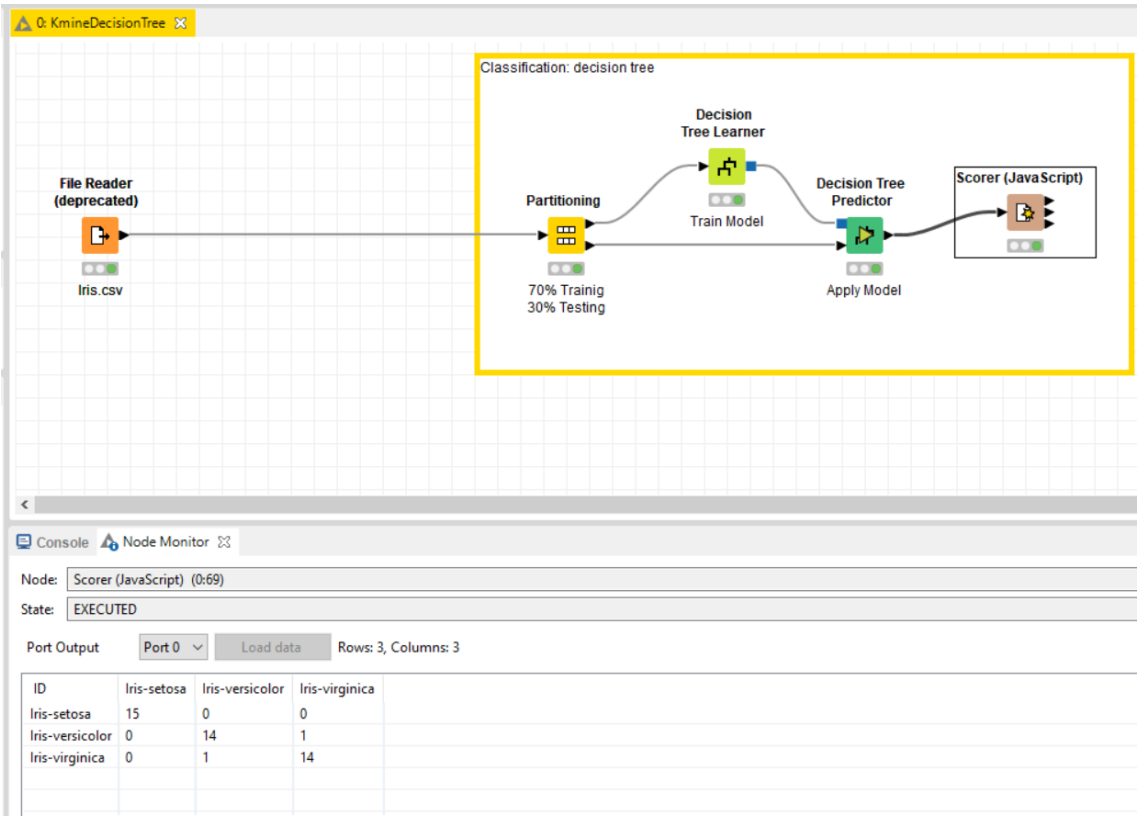
El modo balanced usa valores de y para ajustar los pesos automáticamente inversamente proporcional a la frecuencia de las clases en el input como n_samples / (n_clases * np.bincount(y)). Para problemas multi-output los pesos de cada columna de y se multiplican.

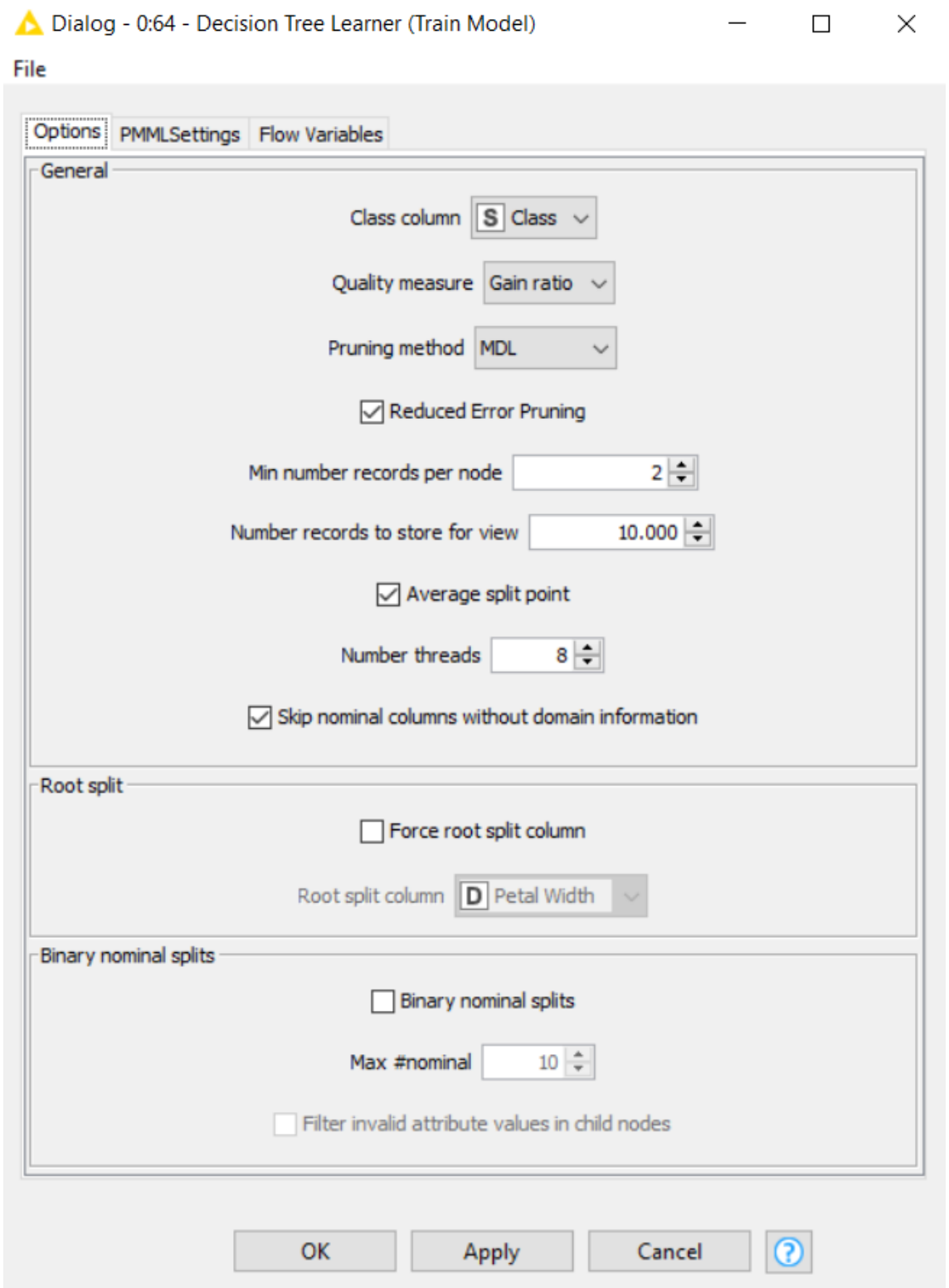
ccp_alphanon-negative float, default=0.0

parámetro de complejidad usado para la poda de complejidad y costo mínimo. El subárbol con el costo de complejidad que es menor que el ccp_alpha va a ser escogido. Por defecto no se hace ninguna poda.

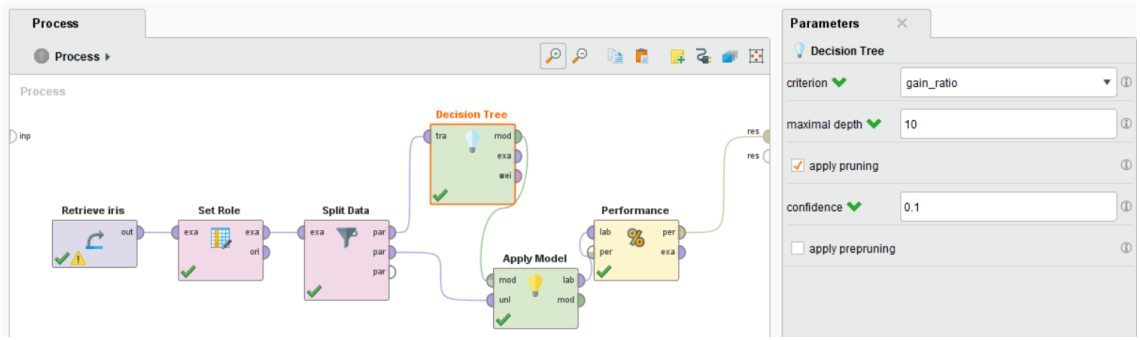
Resultados

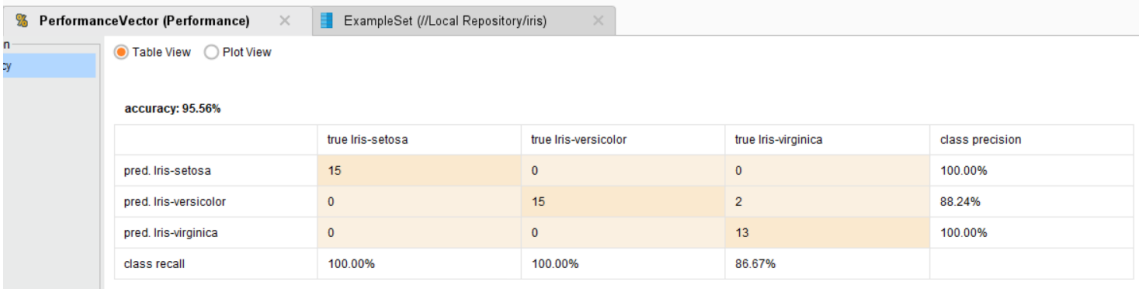
Knime




Dialog - 0:64 - Decision Tree Learner (Train Model)
File
Options
PMMLSettings
Flow Variables
General
Class column
S Class
Quality measure
Gain ratio
Pruning method
MDL
Reduced Error Pruning
Min number records per node
2
Number records to store for view
10.000
Average split point
Number threads
8
Skip nominal columns without domain information
Root split
Force root split column
Root split column
D Petal Width
Binary nominal splits
Binary nominal splits
Max #nominal
10
Filter invalid attribute values in child nodes
OK
Apply
Cancel

Rapid Miner





PerformanceVector (Performance) | ExampleSet (//Local Repository/iris)

Table View | Plot View

accuracy: 95.56%

	true Iris-setosa	true Iris-versicolor	true Iris-virginica	class precision
pred. Iris-setosa	15	0	0	100.00%
pred. Iris-versicolor	0	15	2	88.24%
pred. Iris-virginica	0	0	13	100.00%
class recall	100.00%	100.00%	86.67%	

Conclusiones

Ambas herramientas tienen similaridades en su interfaz gráfica. Hay diferencias en los parámetros que permiten knime no permite pre-prunning por ejemplo, ambos permiten usar Gini y gain ratio, knime presenta una opción muy interesante que no está en rapidMiner que es la de asignar núcleos al trabajo para que haga en paralelo.