

# Clasificación MLP con Optimización de Hiperparámetros

Extenderemos el ejercicio de semana 1 para incluir validación cruzada y optimización de hiperparámetros.

## Diagrama General del Proceso

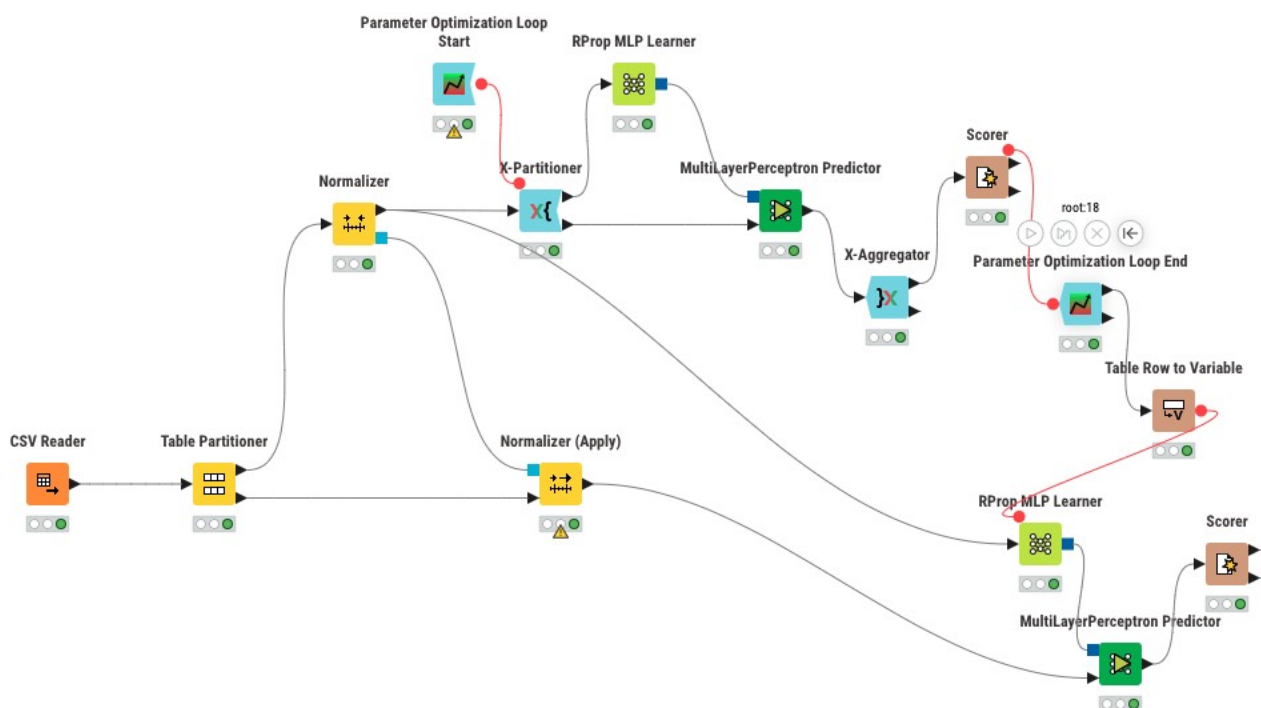
El workflow de KNIME implementa un proceso completo de machine learning con las siguientes etapas:

Datos → Normalizer → Parameter Optimization Loop → Modelo Final → Evaluación

Este flujo extiende el ejercicio básico de la Semana 1 añadiendo dos componentes críticos:

- **Validación cruzada** para evaluar el modelo de forma robusta
- **Optimización de hiperparámetros** para encontrar la mejor configuración

A continuación se muestra el diagrama general del proceso:



## Normalizer (Normalizador)

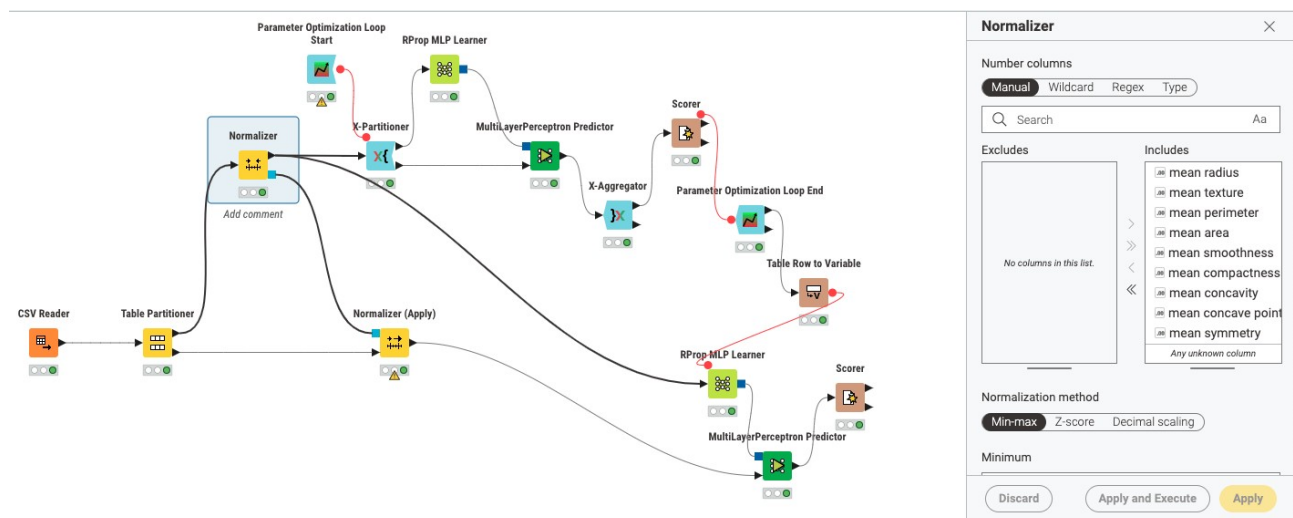
**Función:** Escala los datos antes del entrenamiento

**¿Por qué es importante?**

- Los MLP son sensibles a la escala de las características
- Normaliza todas las variables a un rango similar (típicamente 0-1 o media=0, std=1)
- Debe aplicarse ANTES de cualquier división de datos para evitar data leakage en el loop

## Configuración típica:

- Método: Min-Max (0-1) o Z-Score (estandarización)



## Parameter Optimization Loop Start (Inicio del Loop de Optimización)

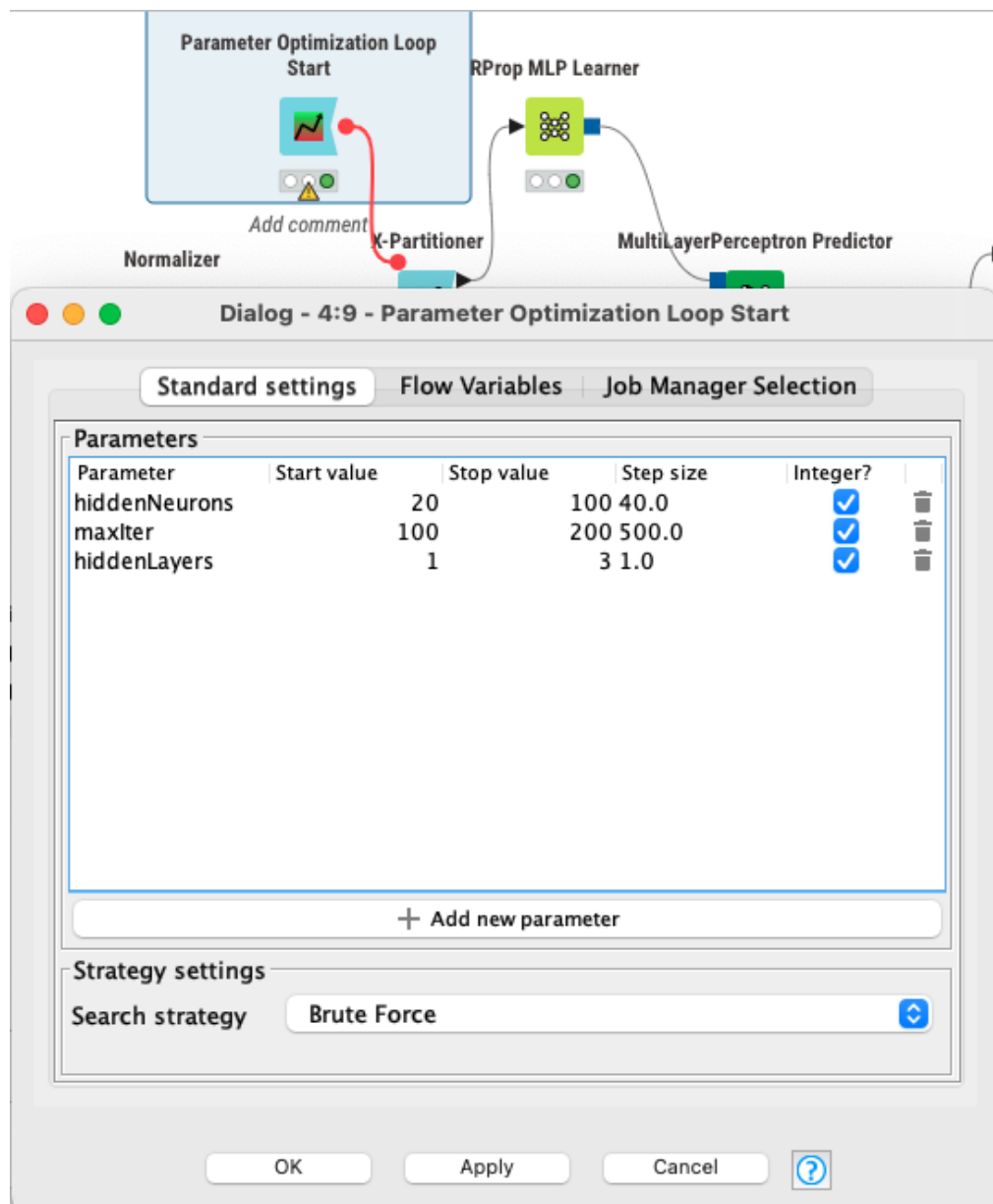
**Función:** Inicia el proceso iterativo de búsqueda de hiperparámetros

### ¿Qué hace?

- Define el **grid de hiperparámetros** a explorar
- Crea combinaciones de parámetros para probar (ej: diferentes arquitecturas de red, tasas de aprendizaje, regularización)
- Cada iteración del loop probará una combinación diferente

### Ejemplo de parámetros a optimizar:

- hiddenNeurons: 20, 60, 100
- maxIter: 100, 300, 500
- hiddenLayers: 1, 2, 3



## X Partitioning (Cross-Validation) (dentro del loop)

**Función:** Implementa la validación cruzada (X-fold)

**¿Cómo funciona?**

- Divide los datos de entrenamiento en X "folds" (típicamente 5 o 10)
- En cada iteración:
  - X-1 folds se usan para entrenar
  - 1 fold se usa para validar
- Esto se repite X veces, rotando el fold de validación

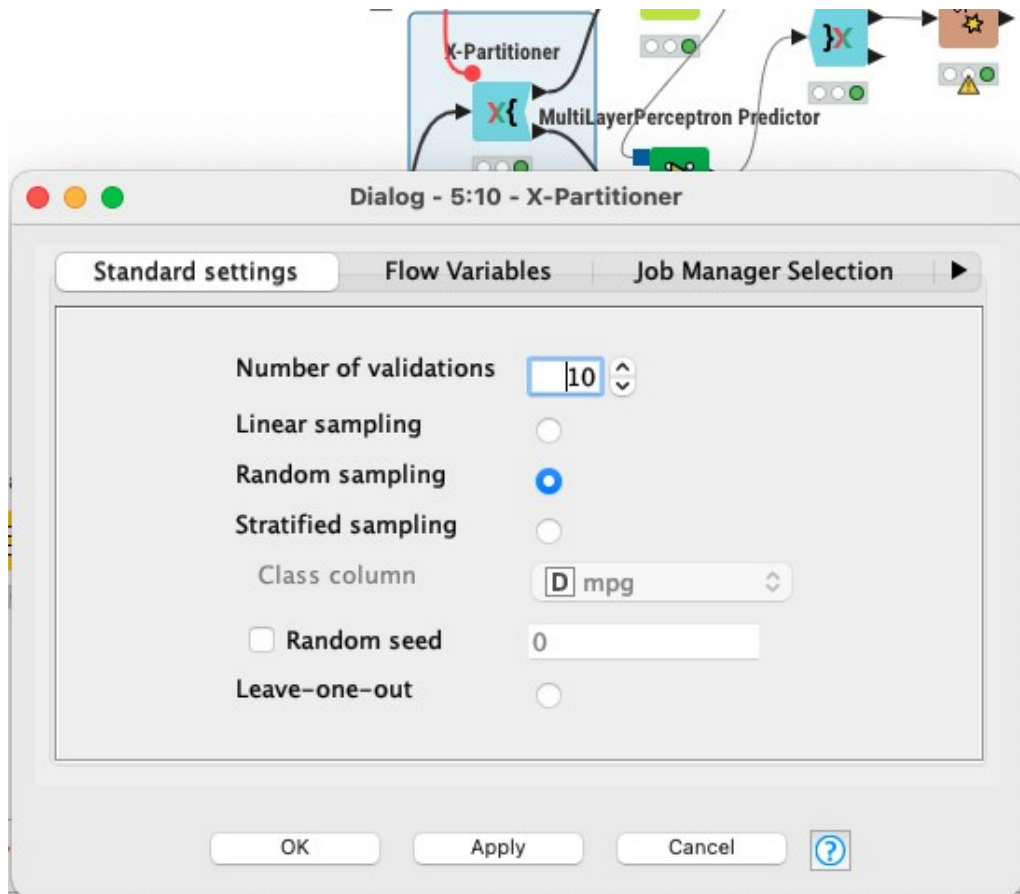
**Ventajas de Cross-Validation:**

- Usa todos los datos tanto para entrenamiento como validación
- Reduce el riesgo de overfitting
- Proporciona una estimación más robusta del rendimiento

- Evita la dependencia de una única división train/test

### En KNIME:

- Este componente está implícito en el loop de optimización
- Cada combinación de hiperparámetros se evalúa con CV

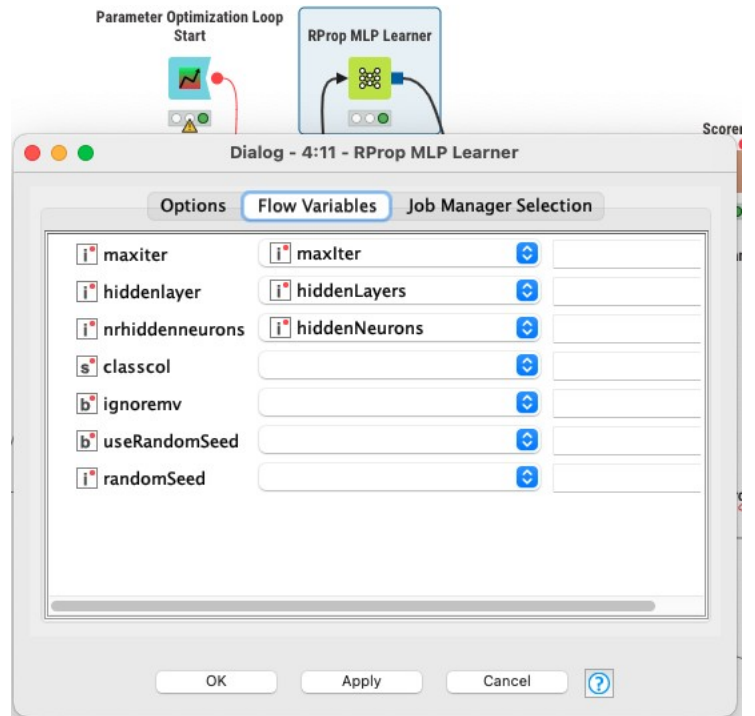
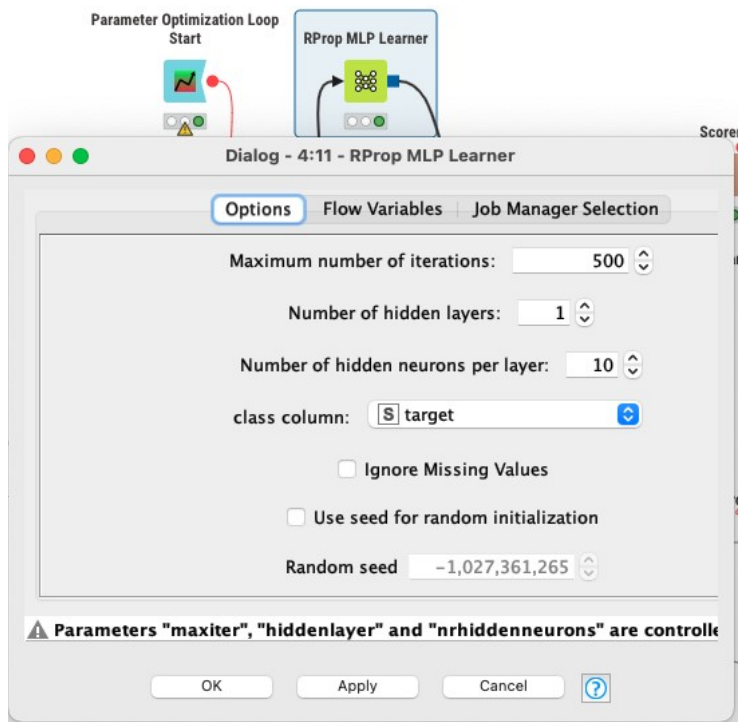


### MLP Learner (dentro del loop)

**Función:** Entrena el modelo de red neuronal con los hiperparámetros actuales

#### ¿Qué hace en cada iteración del loop?

- Recibe una combinación específica de hiperparámetros
- Entrena el MLP en los datos de entrenamiento de cada fold
- Genera un modelo entrenado para evaluación



## X Aggregator

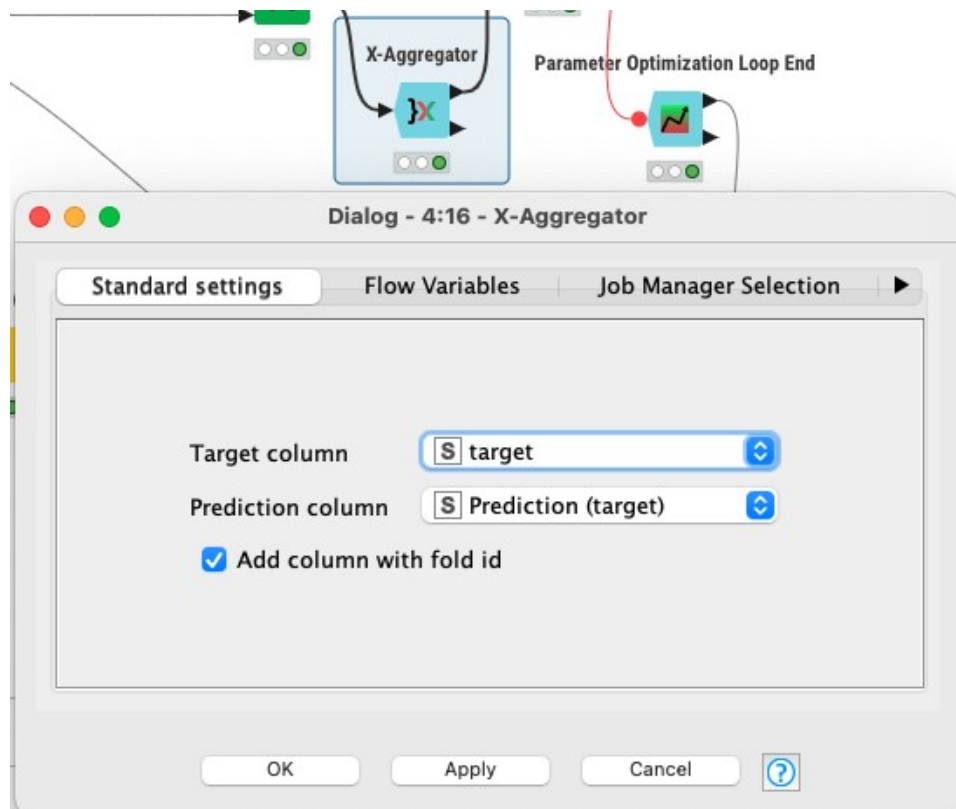
**Función:** Agrega los resultados de todos los folds de validación cruzada

**¿Qué hace?**

- Recopila los scores de cada fold
- Calcula el **score promedio** y **desviación estándar**
- Proporciona una métrica única de rendimiento para la combinación de hiperparámetros actual

**Salida:**

- Score medio de CV (ej: accuracy =  $0.95 \pm 0.02$ )
- Esto representa el rendimiento esperado del modelo



## Scorer (dentro del loop)

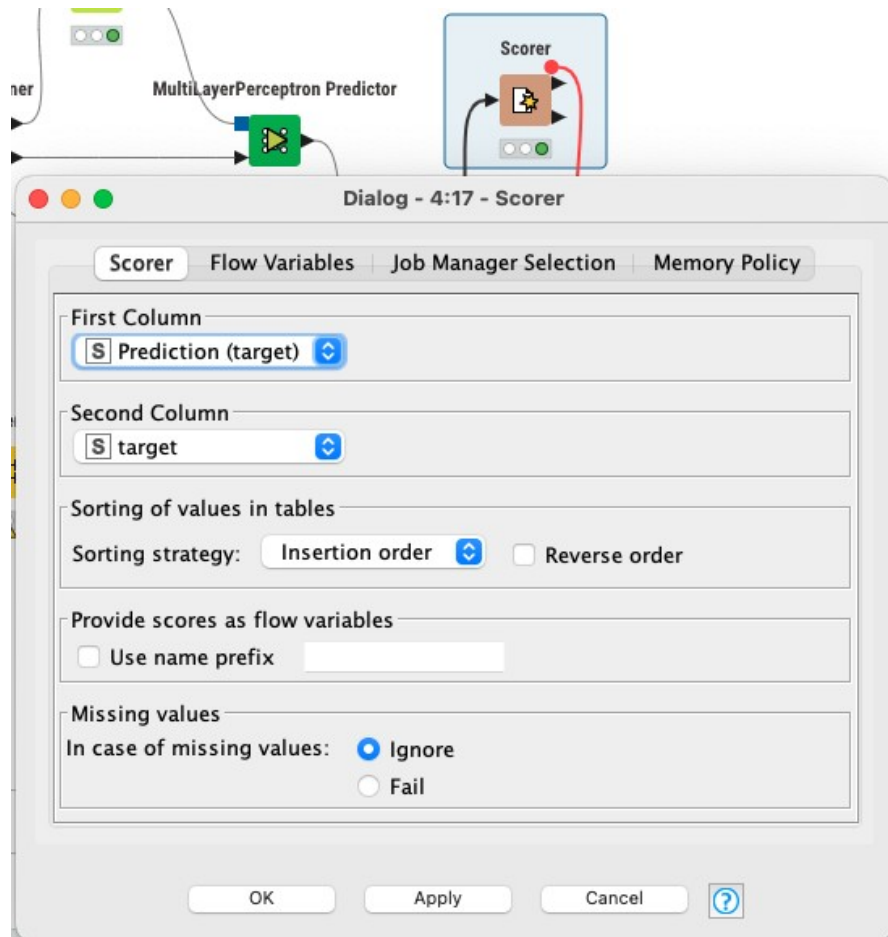
**Función:** Evalúa el rendimiento del modelo en el fold de validación

**Métricas calculadas:**

- **Accuracy:** Porcentaje de predicciones correctas
- **Precision:** Correctitud de las predicciones positivas
- **Recall:** Capacidad de encontrar todos los casos positivos
- **F1-Score:** Media armónica de precision y recall
- **AUC-ROC:** Área bajo la curva ROC

**Propósito:**

- Medir qué tan bien funciona cada combinación de hiperparámetros
- Los scores se promedian a través de todos los folds del CV



## Parameter Optimization Loop End (Fin del Loop)

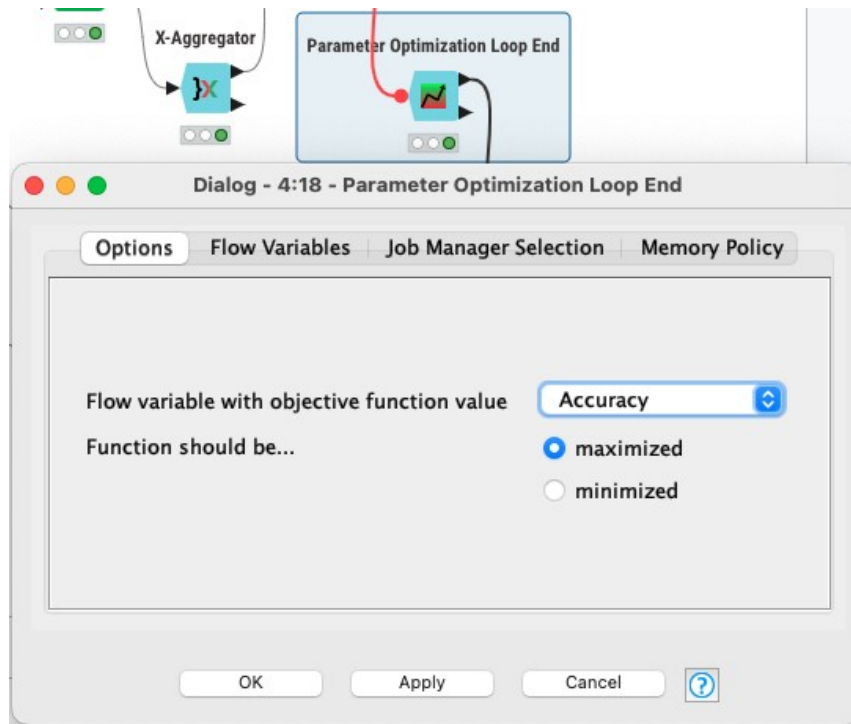
**Función:** Cierra el loop y selecciona los mejores hiperparámetros

**¿Qué hace?**

- Compara todos los scores de las diferentes combinaciones
- Identifica la combinación con el **mejor score promedio de CV**
- Retorna los hiperparámetros óptimos

**Criterio de selección:**

- Mejor accuracy/F1-score en validación cruzada
- En caso de empate, puede preferir modelos más simples (menor complejidad)



## Table Row to Variable

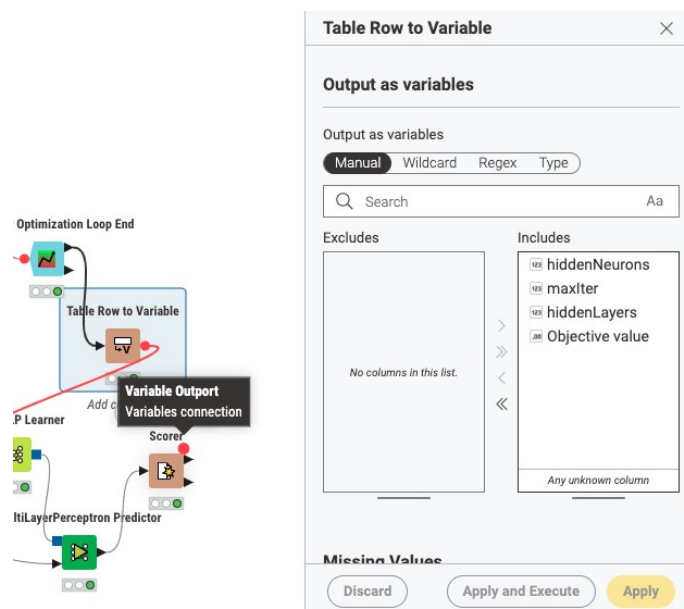
**Función:** Convierte los mejores hiperparámetros en variables

**¿Por qué es necesario?**

- Extrae los valores óptimos de hiperparámetros de la tabla de resultados
- Los convierte en variables de flujo que pueden usarse en nodos posteriores
- Permite configurar automáticamente el modelo final

**Ejemplo:**

- Extrae: `hiddenNeurons = 20`, `maxIter=100`, `hiddenLayers = 2`
- Estas variables se pasan al siguiente MLP Learner





## MLP Learner (Final)

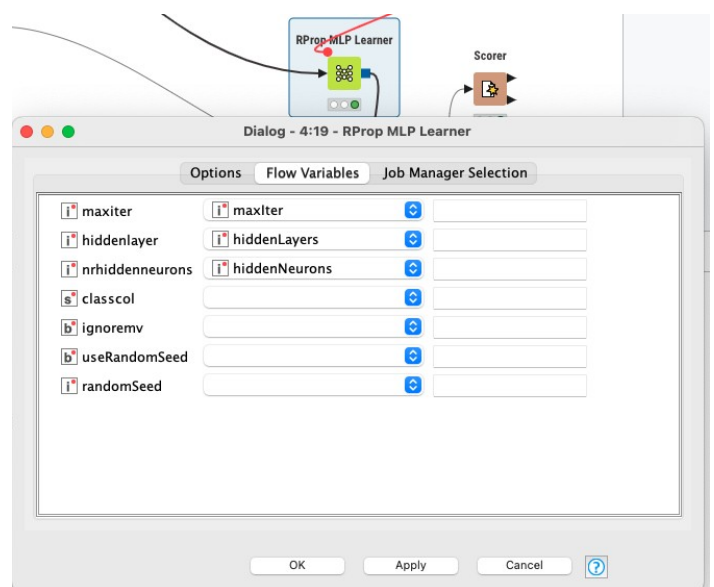
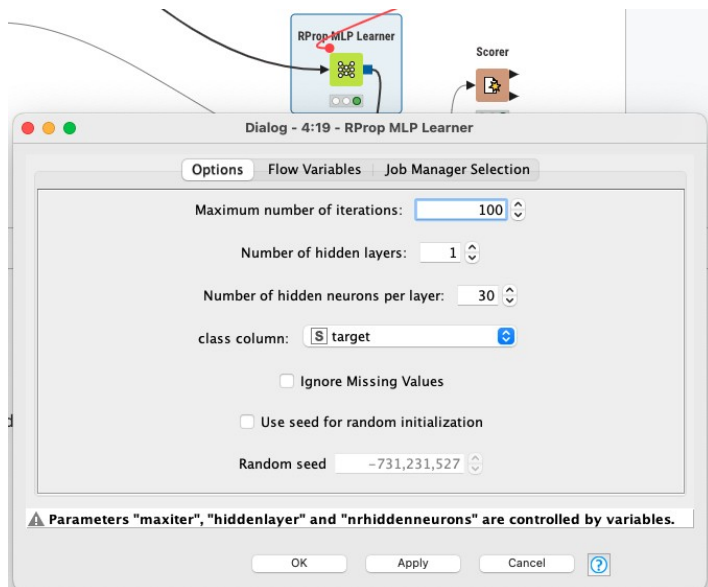
**Función:** Entrena el modelo final con los mejores hiperparámetros

**Diferencia con el MLP del loop:**

- Este se entrena con **TODOS** los datos de entrenamiento (no solo folds)
- Usa los hiperparámetros optimizados encontrados en el loop
- Genera el modelo final que se usará para predicciones

**¿Por qué reentrenar?**

- El loop usó solo subconjuntos de datos (folds)
- El modelo final aprovecha el 100% de los datos de entrenamiento
- Esto maximiza el aprendizaje antes de evaluar en test set



## Scorer (Final)

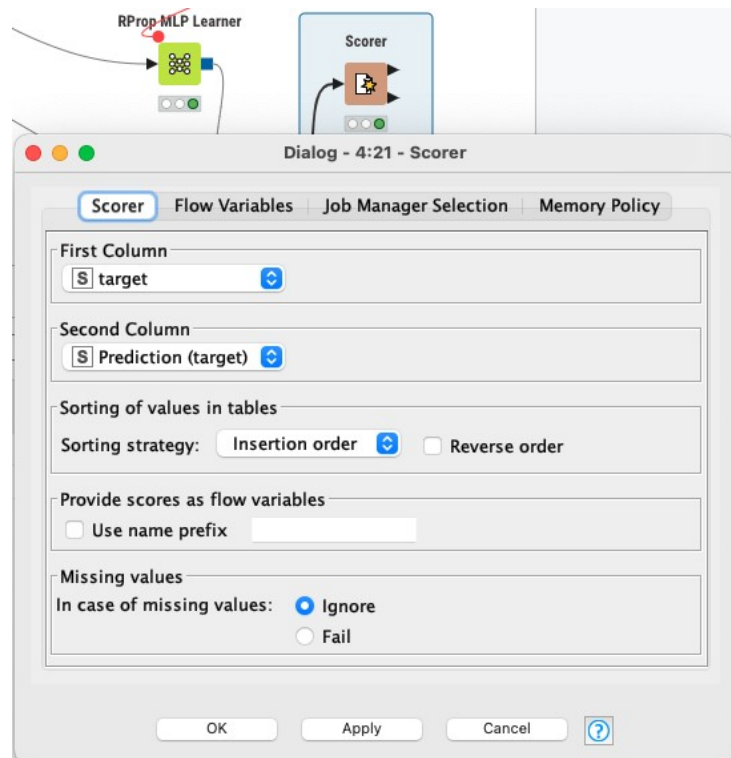
**Función:** Evalúa el modelo final en el conjunto de prueba (test set)

**¿Qué mide?**

- Rendimiento del modelo en datos **completamente no vistos**
- Esta es la métrica que reportas como resultado final
- Incluye accuracy, precision, recall, F1, matriz de confusión

### Importancia:

- Es la evaluación "honesta" del modelo
- El test set nunca se usó durante la optimización
- Refleja el rendimiento esperado en producción



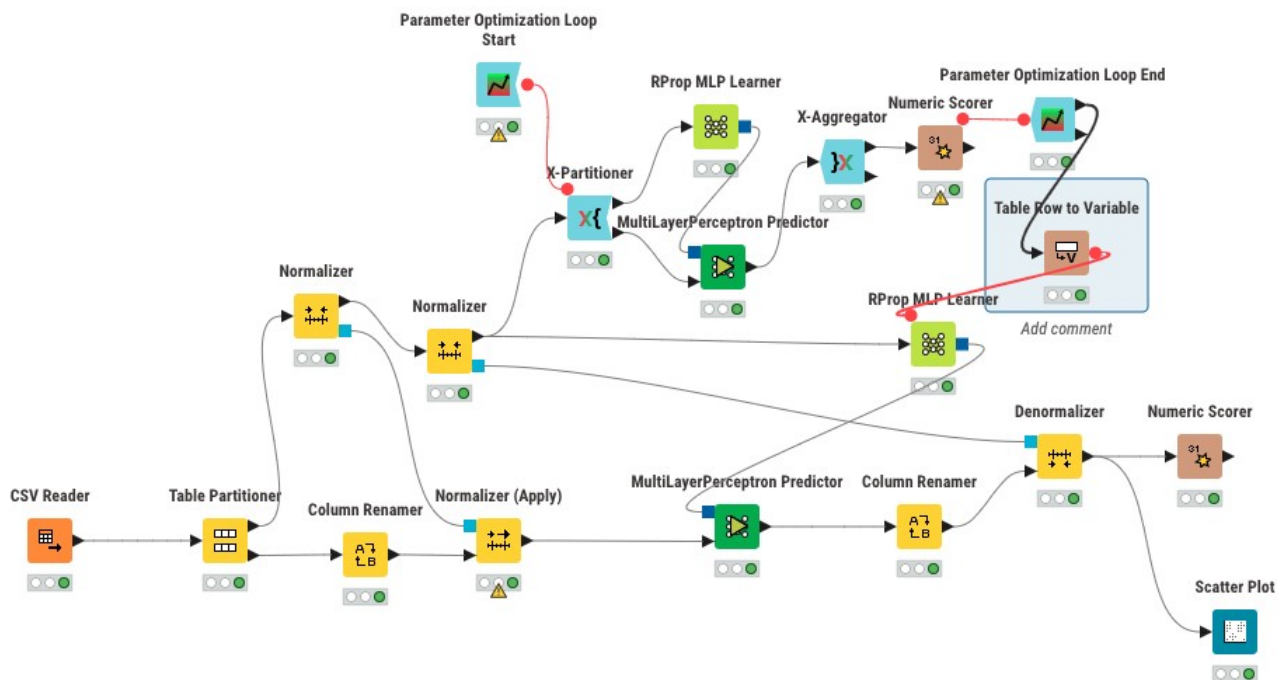
## Flujo de Regresión MLP en KNIME con Optimización de Hiperparámetros

### ● DIFERENCIA CRÍTICA: Regresión vs Clasificación

En regresión con MLP en KNIME, hay una complicación adicional:

⚠ El MLP Learner requiere que tanto las características (X) COMO el target (y) estén escalados

Esto NO ocurre en clasificación (donde el target es categórico), pero en regresión el target es numérico y debe normalizarse para que el MLP converja correctamente.



## Flujo General del Proceso

### Parte 1: Preparación de Datos (Flujo Inferior)

CSV Reader → Table Partitioner → Column Renamer → Normalizer →  
 Normalizer (Apply) → [Datos listos para el loop]

#### ¿Qué sucede aquí?

1. **CSV Reader:** Carga el dataset de regresión
2. **Table Partitioner:** Divide en Training (80%) y Test (20%)
3. **Column Renamer:** Renombra columnas para claridad
4. **Normalizer:**
  - Se ajusta (fit) con los datos de **training**
  - Normaliza TODAS las columnas incluyendo el **target**
  - Guarda los parámetros de normalización (media, std o min/max)
5. **Normalizer (Apply):**
  - Aplica la MISMA transformación al conjunto de **test**
  - Usa los parámetros aprendidos del training
  - **Crítico:** Evita data leakage

#### ¿Por qué dos Normalizers?

- El primero aprende y transforma (training)
- El segundo solo transforma (test) usando los mismos parámetros
- Esto garantiza consistencia entre training y test

### Parte 2: Loop de Optimización (Flujo Superior)

Parameter Optimization Loop Start → X-Partitioner →

RProp MLP Learner → MultiLayerPerceptron Predictor →  
X-Aggregator → Numeric Scorer → Parameter Optimization Loop End

### ¿Qué sucede aquí?


- 1. Parameter Optimization Loop Start:**
  - Define el grid de hiperparámetros a explorar
  - Inicia las iteraciones
- 2. X-Partitioner:**
  - Implementa validación cruzada (5 o 10 folds)
  - Divide el training set en sub-training y validation
- 3. RProp MLP Learner:**
  - Entrena el MLP con los datos **escalados** (incluyendo target escalado)
  - RProp = Resilient Propagation (algoritmo de optimización específico)
  - **Requiere que el target esté normalizado**
- 4. MultiLayerPerceptron Predictor:**
  - Hace predicciones en el fold de validación
  - Las predicciones están en **escala normalizada**
- 5. X-Aggregator:**
  - Agrega resultados de todos los folds del CV
  - Calcula el score promedio
- 6. Numeric Scorer:**
  - Calcula métricas: MSE, RMSE, MAE,  $R^2$
  - **Las métricas se calculan en escala normalizada**
- 7. Parameter Optimization Loop End:**
  - Selecciona los mejores hiperparámetros
  - Termina el loop

### Parte 3: Modelo Final y Evaluación

Table Row to Variable → RProp MLP Learner (Final) →  
MultiLayerPerceptron Predictor → Column Renamer →  
Denormalizer → Numeric Scorer (Final) + Scatter Plot

### ¿Qué sucede aquí? (Lo más importante)

- 1. Table Row to Variable:**
  - Extrae los mejores hiperparámetros del loop
- 2. RProp MLP Learner (Final):**
  - Entrena con TODO el training set normalizado
  - Usa los hiperparámetros optimizados
- 3. MultiLayerPerceptron Predictor:**
  - Predice en el test set
  - **Predicciones en escala normalizada**
- 4. Column Renamer:**

- Organiza las columnas de predicción
5.  **Denormalizer (CRÍTICO):**
- **Revierte la normalización del target**
  - Convierte las predicciones de escala normalizada a escala original
  - Usa los mismos parámetros del Normalizer inicial
  - **Sin esto, las predicciones no tienen sentido en el contexto real**
6. **Numeric Scorer (Final):**
- Calcula métricas finales en **escala original**
  - MSE, RMSE, MAE,  $R^2$  ahora tienen unidades interpretables
7. **Scatter Plot:**
- Visualiza Observados vs Predichos en escala original



## Flujo de Normalización/Desnormalización

### Durante el entrenamiento:

Datos originales → Normalizer → Datos escalados → MLP Learner

### Durante la predicción:

Datos originales → Normalizer (Apply) → Datos escalados →  
MLP Predictor → Predicciones escaladas → Denormalizer →  
Predicciones originales



## Complicaciones Adicionales por el Escalado del Target

### 1. Dos normalizaciones separadas:

- Normalización de **features** (X)
- Normalización del **target** (y)
- Ambas deben ser consistentes entre train/test

### 2. Desnormalización obligatoria:

- Las predicciones del MLP están en escala normalizada
- **No tienen significado real** hasta desnormalizar
- El Denormalizer debe usar los **MISMOS** parámetros del Normalizer original

### 3. Métricas en dos escalas:

- **Dentro del loop:** Métricas en escala normalizada (para comparar modelos)
- **Evaluación final:** Métricas en escala original (para interpretar resultados)

#### 4. Más nodos en el workflow:

- Clasificación: ~8 nodos
- Regresión: ~12+ nodos (por la normalización extra)



### Resumen: Clasificación vs Regresión en KNIME

Aspecto	Clasificación	Regresión
Target	Categorico (no se escala)	Numérico (DEBE escalarse)
Normalización	Solo features	Features + Target
Desnormalización	No necesaria	<b>OBLIGATORIA</b>
Complejidad del workflow	Media	Alta
Métricas	Accuracy, F1, Confusion Matrix	MSE, RMSE, MAE, R <sup>2</sup>
Visualización final	Confusion Matrix Heatmap	Scatter Plot (Obs vs Pred)

---



### Concepto Clave

En regresión con MLP, el workflow se complica porque debes:

1. Normalizar el target antes del entrenamiento
2. Entrenar el MLP en datos completamente normalizados
3. Desnormalizar las predicciones antes de evaluarlas
4. Mantener consistencia de parámetros entre train/test

**Todo esto para que el MLP pueda aprender eficientemente en una escala numérica estable, pero los resultados finales sean interpretables en la escala original de los datos.**

---



### Lección Práctica

Si tus predicciones de un MLP de regresión parecen estar en un rango extraño (ej: predices precios de casas y obtienes valores entre 0 y 1), probablemente **olvidaste desnormalizar** el target.

Este es uno de los errores más comunes en regresión con redes neuronales en KNIME.