DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Sistemas Inteligentes de Gestión

Guión de Prácticas de Minería de Datos

Práctica 4

Clasificación y regresión

© *Juan Carlos Cubero & Fernando Berzal*

FICHEROS DE DATOS

```
Datos de empleados.sav
Mundo 95.sav
agaricus-lepiota.csv
iris.csv
credit-german.arff
```

ENTREGA DE LA PRÁCTICA

```
Regresión.doc
Regresión.spo

Clasificación.doc
Clasificación_Mushroom.knime.zip
Clasificación_Iris.knime.zip
Clasificación_Credit.knime.zip
```

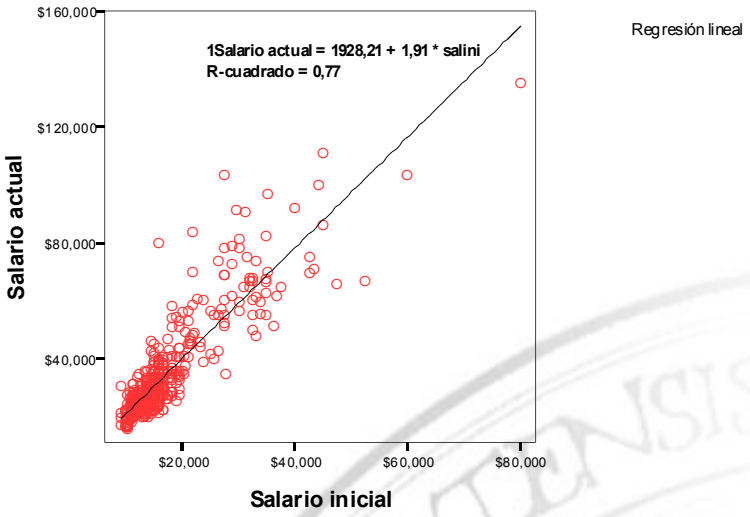
Ejercicios tipo C: Regresión

Datos de empleados

Ejecute SPSS y abra el fichero "Datos de empleados".

En *Gráficos* > *Interactivos* > *Diagrama de dispersión*, seleccione *Salario actual* como variable dependiente y *Salario Inicial* como variable independiente.

En la pestaña *Ajuste*, seleccione *Regresión* en la lista desplegable asociada a *Método* y marque la casilla que indica "*Incluir constante en la ecuación*".



Con esto podemos comprobar gráficamente si tiene sentido plantear un modelo de regresión lineal (es decir, si se aprecia, aunque sea aproximadamente, una relación lineal entre las dos variables). El visor de resultados de SPSS también nos muestra el coeficiente de correlación al cuadrado (*R-cuadrado*)

Guarde el contenido del visor en el fichero `Regresión.spo`, copie el gráfico obtenido e inclúyalo en su fichero `Regresión.doc`, junto con su análisis de los resultados obtenidos.

Mundo 95

A continuación, sobre el fichero de datos `Mundo 95.sav`, construya un diagrama de dispersión y cree un modelo de regresión lineal utilizando *Mortalidad Infantil* como variable dependiente e *Ingesta diaria de calorías* como variable independiente. No olvide guardar el contenido del visor SPSS en el fichero `Regresión.spo`, copiar el gráfico obtenido en `Regresión.doc` e incluir su interpretación de los resultados obtenidos.

Ejercicios tipo C: Clasificación (manual)

Construya un conjunto de datos artificial que consistirá en una tabla con cinco atributos (A, B, C, D, CLASE) sujetos a las siguientes restricciones:

- Dominio(A) = {a1, a2, a3}
- Dominio(B) = {b1, b2, b3, b4}
- Dominio(C) = {c1, c2}
- Dominio(D) = {d1, d2, d3, d4, d5}
- Dominio(CLASE) = {clase1, clase2, clase3}
- La tabla debe incluir, al menos, 15 tuplas.

MUY IMPORTANTE: Cada alumno deberá trabajar con sus propios datos. No se admitirán dos prácticas con los mismos conjuntos de datos.

Construya un árbol de decisión para clasificar los valores de la clase. Como regla de división, se utilizará la entropía. Como criterio de parada, un nodo del árbol se cerrará cuando cubra dos tuplas o menos. Incluya, en un fichero `Clasificación.doc`, los cómputos de todas las entropías correspondientes a los distintos nodos del árbol.

A continuación, construya otra tabla con 5 tuplas a modo de conjunto de prueba y complete la matriz de contingencia que muestre los errores de clasificación que se haya podido cometer.

Incluya los conjuntos de datos y todos los cómputos realizados en `Clasificacion.doc`

Ejercicios tipo C: Mushroom

A partir de los `datos agaricus-lepiota.csv`, construiremos un árbol de decisión que nos ayude a decidir si una seta venenosa (*e-edible*) o no (*p-poisonous*) en función de sus características morfológicas. Para ello, creamos un proyecto en KNIME con los siguientes nodos:

- *Data Manipulation* > *Row* > *Partitioning*

  Para generar el conjunto de entrenamiento y de prueba. Configúrelo para que el primero corresponda al **80%** de los datos (por defecto, aparece un valor muy bajo del 10%) y que la selección sea "estratificada". Esto significa que la distribución de la clase (tanto por ciento de cada uno de los valores de la clase) se mantendrá en cada partición.

  NOTA: El atributo que indica la clase es "*Class*".

- *Mining* > *Decision Tree* > *Decision Tree Learner*

  Para generar el árbol de decisión. Configúrelo utilizando con "*Gain Ratio*" como *Quality Measure* y "*No Pruning*" (sin poda) en *Pruning Method*.

- *Mining* > *Decision Tree* > *Decision Tree Predictor*

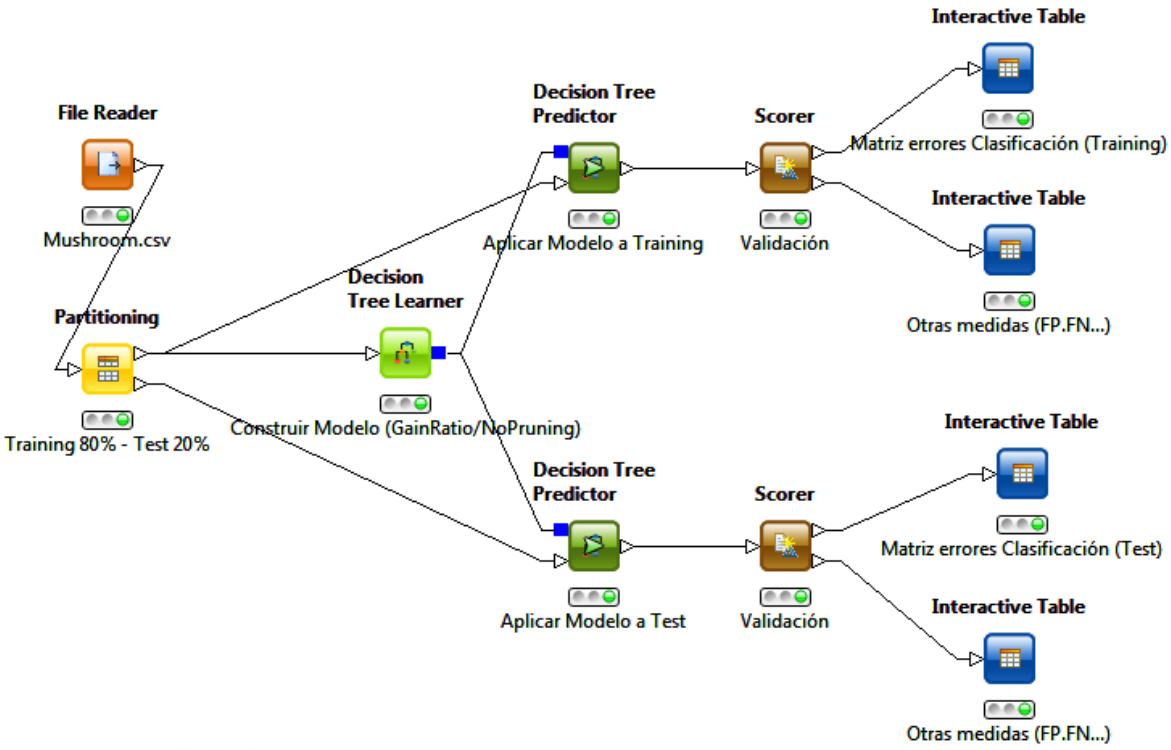
  Para aplicar el árbol obtenido a un conjunto de datos (éste puede ser el mismo conjunto de entrenamiento o bien, lo más usual, el conjunto de prueba). En este ejercicio, utilizaremos ambos.

- *Mining* > *Scoring* > *Scorer*

  Para generar la matriz de contingencia en la que comprobaremos los errores de clasificación. Seleccione las columnas "*Class*" y "*Prediction(DecTree)*" (esta última es la generada por el árbol de decisión utilizado en el nodo anterior, la predicción).

  NOTA: Si hubiésemos usado un nodo Weka para construir el árbol de decisión, habría que seleccionar "*Winner*" en vez de "*Prediction(DecTree)*".

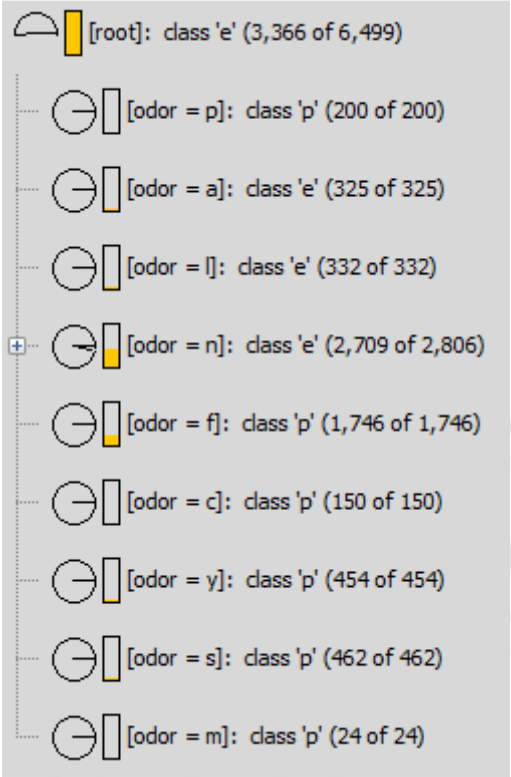
  En el nodo *Scorer*, si pinchamos con la derecha y seleccionamos *View Confusion Matrix*, veremos la matriz de contingencia que muestra los errores de clasificación. Ésta es la primera salida del nodo *Scorer*. La otra salida contiene información adicional sobre otras medidas de ajuste (entre las que se encuentra *Accuracy*; esto es, la precisión del modelo de clasificación).



Tal como hemos configurado nuestro proyecto, usaremos el 80% de los datos como conjunto de entrenamiento para construir el modelo y el 20% restante como conjunto de prueba para validarlo.

Con nuestros datos de setas, la clasificación es perfecta y no hay ningún error (ni en el conjunto de entrenamiento, algo que se podía esperar tras no utilizar técnicas de poda, ni tampoco en el conjunto de prueba). No obstante, esto no será lo usual. Seleccione ahora que se realice una poda del árbol y observe que, al podar algunas de las ramas, se cometen algunos errores.

En *Decision Tree Learner*, podemos ver el árbol construido utilizando la opción *View: Decision Tree View* de su menú contextual.



El icono + indica que se trata de un nodo intermedio que se puede expandir haciendo click sobre él. En los nodos del árbol, se nos muestra la clase más frecuente y cuántos de los casos corresponden a ella del total de nodos cubiertos por el nodo (p.ej. 2709 casos de un total de 2806 en el nodo intermedio de la figura).

Además, el "diagrama de sectores" nos indica gráficamente la proporción de casos de la clase mayoritaria (2709/2806) y la barra naranja nos da la proporción de ejemplos que caen en cada nodo (con respecto a los ejemplos cubiertos por el nodo padre).

Incluya el árbol obtenido (usando técnicas de poda) y las matrices de contingencia que se obtienen en el fichero `Clasificacion.doc`. Comente los resultados obtenidos.

A continuación, utilizaremos Weka para construir nuestro árbol de decisión, para lo cual añadiremos los siguientes nodos a nuestro proyecto KNIME:

- *Weka > Classification Algorithms > Trees > J48*

  J48 es la implementación de C4.5 en Weka (también disponible desde *Mining > Classification > Decision Tree > J48(Weka)*). En las opciones de configuración del nodo, asegúrese de que se poda el árbol (`unpruned = false`).

- *Weka > Predictors > Weka Predictor*

  Para generar la matriz de errores de clasificación. Seleccione la columna que representa la clase en el conjunto de datos original y "*Winner*", que es la columna que indica la predicción realizada por el modelo creado por Weka.

El árbol generado con el nodo J48 puede mostrarse en formato gráfico (*Graph*) o en texto (*Weka Output*). No obstante, el formato gráfico es propio de Weka y sólo es útil cuando hay pocos nodos, ya que cuando hay muchos, éstos no se muestran adecuadamente. Copie el texto correspondiente al árbol generado en el fichero `Clasificación.doc` y realice un análisis similar al de antes.

Ejercicios tipo B: Iris

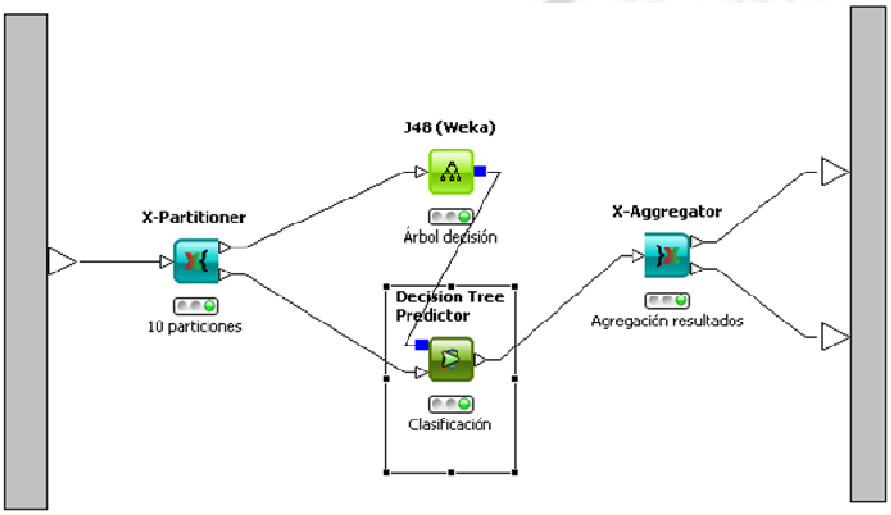
A continuación, crearemos un nuevo proyecto KNIME para crear un modelo de clasificación para el conjunto de datos `Iris`, que ya utilizamos en la práctica de clustering. En este conjunto de datos, el atributo *Class* indica la clase.

Inicialmente, utilizaremos una partición del 80% para el conjunto de entrenamiento y del 20% para el conjunto de test. Construya un árbol de decisión (sin podar) utilizando los nodos de Weka, incluya el árbol creado en `Clasificación.doc` y tradúzcalo manualmente a un conjunto de reglas con antecedentes mutuamente excluyentes.

Observe que hay atributos numéricos que se repiten en distintos niveles del árbol. Modifique las reglas anteriores para que aparezcan divisiones por intervalos en vez de divisiones binarias.

Añada al fichero `Clasificación.doc` la matriz de errores de clasificación obtenida al utilizar el árbol sobre el conjunto de test, así como el valor asociado de *Accuracy*, y comente los resultados obtenidos.

Por último, veamos cómo realizar una validación cruzada. Para ello, añada un nodo de tipo *Meta* > *X-Validation* a su proyecto. Tras añadirlo, haga doble click sobre él y se abrirá un cuadro en el que podrá definir el flujo interno de este meta-nodo:



Tras finalizar la definición de este meta-nodo, conectamos su salida superior a un *Scorer* y su salida inferior a una tabla interactiva. Copie los resultados de dichos nodos en `Clasificacion.doc` y compárelos con los obtenidos validación cruzada.

En las transparencias finales de clase se mencionan otros tipos de clasificadores, como el de los k-vecinos más cercanos, disponible en KNIME a través de *Mining* > *Misc Classifiers* > *K Nearest Neighbor*. Discuta si tiene sentido aplicar este modelo de clasificación al conjunto de datos `Iris`. En caso afirmativo, hágalo con k=5, evalúelo con un nodo *Scorer* y comente los resultados obtenidos en `Clasificación.doc`.

Ejercicios tipo A: Credit-German

Con este ejercicio vamos a estudiar una aplicación real muy utilizada en el sector bancario. Se trata de medir el riesgo asociado a la concesión de un crédito bancario [*credit scoring* en inglés].

Utilizaremos el fichero de datos credit-german.arff y crearemos un nuevo proyecto en KNIME. El fichero de datos aparece descrito en:

http://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)

El fichero contiene 1000 ejemplos que representan clientes de una entidad bancaria que demandaron un crédito, para cada uno de los cuales se recopilaron 7 atributos numéricos y 13 nominales. Los atributos indican información sobre el cliente en cuestión (estado civil, edad, parado...) y sobre el crédito solicitado (propósito del crédito, cantidad solicitada, etc.). El atributo que indica la clase ["*Class*"] es binario e indica si el cliente puede ser considerado fiable a la hora de concederle el crédito [*good*] o no [*bad*].

Construya un árbol de clasificación con una partición del 80% para el conjunto de entrenamiento y del 20% para el conjunto de prueba. Utilice un nodo de tipo *Weka > Classification Algorithms > Trees >J48*. Copie el árbol de clasificación obtenido en el fichero `Clasificación.doc`, indicando la precisión obtenida [*Accuracy*], que estará en torno al 65%, dependiendo del conjunto de entrenamiento seleccionado.

Añada ahora un nodo de tipo *Statistics >Value Counter*, enlazado a la fuente de datos, para contar cuantas tuplas hay de cada clase. Verá que un 70% de los clientes clasificados corresponden a la clase "*good*". Por tanto, el clasificador más básico de todos, que sería aquel que siempre clasifica con la clase más frecuente, ¡tendría una mayor precisión que el complejo árbol de decisión que habíamos obtenido!. Esto nos muestra que no siempre será posible obtener un buen modelo de clasificación usando un único método (en nuestro caso, árboles de decisión).

Existen multitud de métodos alternativos y empezaremos utilizando un modelo de clasificación basado en listas de decisión de la siguiente forma:

```
si a entonces c1
si no
   si b entonces c2
   si no
      si h entonces c1
      si no
         c2
```

Para ahorrarnos la repetición de tantos "`si no`", la lista de decisión la mostraremos de la siguiente forma:

```
si a entonces c1
si b entonces c2
si h entonces c1
c2
```

Al lado de cada regla de la lista de decisión, se suele añadir un par de valores `(t1, t2)` donde `t1` indica el número de tuplas cubiertas por la regla y `t2` el número de errores cometidos.

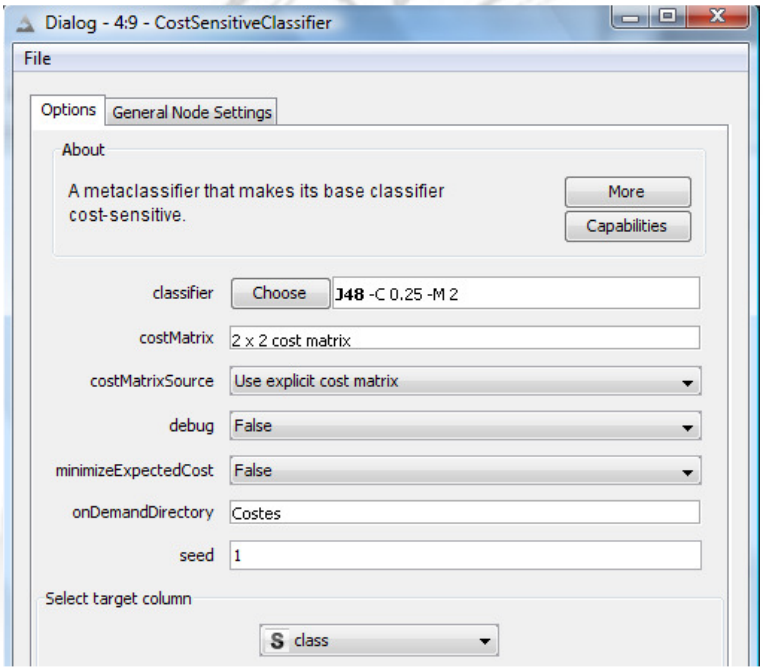
Veámoslo en la práctica añadiendo a nuestro proyecto KNIME un nodo de tipo *Weka* > *Classification Algorithms* > *Rules* > *JRip*, que implementa el método Ripper, e incluya el resultado en `Clasificación.doc`.

Compare los resultados obtenidos en términos de complejidad (número de reglas) y precisión con respecto al árbol de decisión anterior.

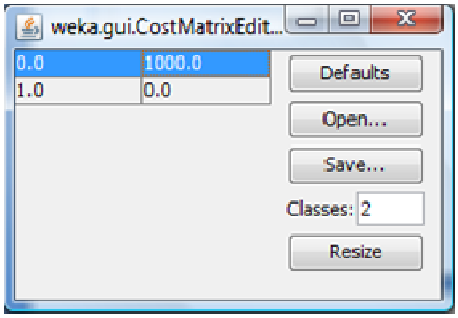
> NOTA: Para mejorar los resultados de precisión en estos ejemplos, puele realizarse una clasificación por grupos. Primero se forman varios grupos de clientes, atendiendo a sus características (aplicando, por ejemplo, técnicas de clustering) y, para cada grupo, se obtiene un modelo de clasificación distinto. Para nuestro ejemplo concreto, puede consultarse el siguiente trabajo a modo de ampliación: http://itc.ktu.lt/itc361/Zakzewska361.pdf

La clasificación errónea de un cliente como malo, cuando realmente es bueno, tiene un coste para el banco mucho menor que la clasificación de un cliente como bueno, cuando realmente es malo. Para incluir esta restricción en la construcción de nuestro modelo de clasificación, debemos utilizar costes. Los modelos que tienen en cuenta estos costes relativos se conocen con el término *cost-sensitive* en inglés.

KNIME no proporciona ningún método sensible a los costes para clasificar, pero podemos utilizar un nodo de tipo *Weka* > *Meta* > *CostSensitiveClassifier*. Este proporciona un mecanismo general mediante el que aplicar costes a cualquier modelo de clasificación (duplicando artificialmente el número de tuplas con un valor concreto de clase, en función de los costes de clasificación de dicha clase).



Podemos especificar J4.8 como modelo de clasificación e introducir manualmente nuestra matriz de costes:



Implícitamente, las clases se ordenan alfabéticamente (por lo que la primera fila corresponde a bad y la segunda a good). Las filas indican la clase real y las columnas la predicción realizada por el modelo (en nuestro caso, es 1000 veces peor concederle un crédito a alguien poco fiable que no concedérselo a alguien que podría devolvérnoslo).

La opción *minimizeExpectedCost* utiliza otro método de aplicación de costes, pero éste sólo es válido para clasificadores probabilísticos, de ahí que no lo utilicemos.

¡CUIDADO! Cuando en la opción *costMatrixSource* indicamos que se lea de un fichero, KNIME no lo recuerda, por lo que si cambiamos la matriz del fichero, hay que seleccionar de nuevo la opción *costMatrixSource* > *Load cost matrix on demand* para que vuelva a leer los cambios realizados en la matriz.

En cualquier caso, nosotros introduciremos manualmente nuestra matriz de costes, de tal forma que los falsos positivos sean 1000 veces peores que los falsos negativos.

También crearemos otro modelo cambiando 1000 por 5.

Incluya en `Clasificación.doc` las matrices de errores de clasificación de ambos clasificadores cost-sensitive y comente los resultados obtenidos. Compárelos también con el modelo de clasificación obtenido sin utilizar costes. Incluya su discusión en el fichero `Clasificación.doc`.

EVALUACIÓN DE LAS PRÁCTICAS

Los ficheros `Clasificación.doc` y `Regresión.doc` han de incluir todas sus respuestas a las distintas preguntas que se formulan en este guión.

PD: No olvide incluir también sus proyectos KNIME y el fichero correspondiente a su sesión en SPSS.