Técnicas escalables de análisis de datos en entornos Big Data: Clasificadores

Inducción de árboles de decisión en *ML*

Contenido

1. Árboles de decisión en *ML.*
2. Parámetros.
3. Ejemplo: datos car y preparación de datos.
4. Inducción de árboles y ejercicios.
5. OneHotEncoder y árboles de decisión.
6. Referencias.

1. Árboles de decisión en *ML*

- *ML* incluye clases para inducir **árboles de decisión binarios**, con atributos continuos y/o discretos.
- Clasificación binaria o multiclase.
- Regresión.

```
import org.apache.spark.classification.DecisionTreeClasifier
```

- `DecisionTreeClassifier` es un `estimator`.
- Se crea una instancia y se parametriza.
- Se entrena con `fit` para generar un. `DecisionTreeClassifierModel`.

- Se aplica con `transform`.

2. Parámetros de entrada/salida

- Entrada: columnas del Data Frame para entrenar/predecir.
- `labelCol`, por defecto "`label`", tipo `Double`.
- `FeaturesCol`, por defecto "`features`", tipo `Vector`.

- Salida: columnas del nuevo Data Frame al aplicar `transform`.
- `PredictionCol`, por defecto "`prediction`", de tipo `Double`
  - La clase predicha.
- `rawPredictionCol` por defecto "`rawPrediction`", de tipo `Vector`
  - Vector con el número de instancias de entrenamiento de cada clase del nodo hoja que hace la predicción.
- `probabilityCol`, por defecto "`probability`", de tipo `Vector`
  - A partir del vector anterior, normalizando.

- Mejor dejar los nombres por defecto.
- Comunes a la mayoría de los algoritmos.

Parámetros del algoritmo

- `impurity:` Medida de impureza para seleccionar los atributos.
- `maxDepth:` Profundidad máxima del árbol.
- `maxBins:` Número máximo de particiones de atributos continuos.
- `minInstancesPerNode:` número mínimo de instancias en un nodo hijo para realizar una partición.

- Los tres últimos influyen notablemente en el sobreajuste.

Ejemplo

```
/* Creamos una instancia de DecisionTreeClassifier  */
val DTcar=new DecisionTreeClassifier()

/* Elegimos parámetros del modelo        */
val impureza =  "entropy"
val maxProf = 3
val maxBins =5

/* Fijamos parámetros del modelo        */
DTcar.setImpurity(impureza)
DTcar.setMaxDepth(maxProf)
DTcar.setMaxBins(maxBins)
```

- Alternativa:

```
val DTcar=new
DecisionTreeClassifier().setImpurity(impureza).setMaxDepth(max
Prof).setMaxBins(maxBins)
```

impurity

- Los atributos se seleccionan según la **ganancia de información**.
- Clasificación: dos medidas de impureza, gini, entropy.

- Entropy: $H(D)=-\sum_{i=1}^{n} f_i \log_2 f_i$ — ID3, C4.5
- Índice de gini: $I_G(D)= -\sum_{i=1}^{n} f_i \ (1 - f_i)$ — CART

- En principio:
  - "`entropy`" atributos categóricos.
  - "`gini`" numéricos.

- En la práctica, muy poca diferencia.

maxDepth

- Es la profundidad máxima que puede alcanzar el árbol.
- Un árbol de profundidad 0 tiene un único nodo hoja.
- Un árbol de profundidad 1 tiene un nodo interno y dos nodos hojas.
- Valor por defecto: 5

- Es un **parámetro crítico** que hay que ajustar en la etapa de selección de modelos.

- Compromiso:
  - Árbol más profundo: puede sobreajustar.
  - Árbol menos profundo: puede no incluir un atributo necesario.

- Mejor solución: podar después de crear árbol de prof. máxima, pero no escala bien. *ML*, *Mllib* no lo soportan.

maxBins

- Se consideran los valores de los atributos continuos como posibles umbrales para “discretizar” cada atributo.
- Siempre binario: posibles particiones a más profundidad, distintas en cada rama.

- También para los categóricos si tienen más de 2 valores: $(2^{M-1} -1)$ si M valores. Se agrupan 1 frente al resto:
  - azul | rojo, verde azul, rojo | verde rojo| azul, verde

- Valor máximo: 32 (o el número de instancias)
  - Valor por defecto: 32.
  - Valores elevados favorecen el sobreajuste.
  - Inicialmente, mejor probar un valor bajo, 3, 5 o 7, por ejemplo.
- Como mínimo: número máximo de valores de un atributo categórico - 1 (en clasificación binaria).

`minInstancesPerNode`

- Cada atributo seleccionado induce una partición sobre la parte del conjunto de entrenamiento que llega al nodo.
- Detiene la construcción de la rama si ningún atributo genera particiones con menos de `minInstancesPerNode` en cada rama.
- Valor por defecto: 1.
- No suele ser interesante generar particiones con solo dos instancias.
- Valores pequeños favorecen el sobreajuste.

Parada

- Utiliza tres criterios de parada.

- La profundidad del nodo es igual al parámetro `maxDepth (5)`.
- Ningún candidato mejora la ganancia de información sobre el parámetro `minInfoGain (0)`.
- Ningún candidato produce un nodo con al menos `minInstancesPerNode` (1).

3. Ejemplo: datos car y preparación de datos

- https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

- Tarea de clasificación.
- Clase: evaluación de un coche.
- Posibles valores: unacc, acc, good, vgood.
- Atributos: 6, discretos.
- coste:_compra: vhigh, high, med, low.
  coste:_mantenimiento: vhigh, high, med, low.
  puerta: 2, 3, 4, 5more.
  personas: 2, 4, more.
  maletero: small, med, big.
  seguridad: low, med, high.
- Valores ausentes: ninguno.

Distribución de clases

- Class Distribution (number of instances per class)

Class	N	N[%]
Unacc	1210	(70.023 %)
acc	384	(22.222 %)
good	69	(3.993 %)
v-good	65	(3.762 %)

Procesamos conjunto de datos

- Como en script "`creacionModelosClasificacionML.scala`"
- Recordar: Lectura a Data Frame, `StringIndexer`, `OneHotEncoder`, `VectorAssembler`, `StringIndexer` para la clase
- En torno a la línea 145:

```
// Creamos el DataFrame carFeaturesLabelDF con columnas features y label
val carFeaturesLabelDF = indiceClase.fit(carFeaturesClaseDF).transform(carFeaturesClaseDF).drop("clase")
carFeaturesLabelDF.show(10)
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|(15,[0,3,11,14],[...|  1.0|
|(15,[0,3,11,13],[...|  1.0|
|(15,[0,3,11],[1.0...|  1.0|
|(15,[0,3,12,14],[...|  1.0|
|(15,[0,3,12,13],[...|  1.0|
|(15,[0,3,12],[1.0...|  1.0|
```

Partición aleatorio entrenamiento y prueba

```
/* Realizamos una partición aleatoria de los datos                */
/*   66% para entrenamiento, 34% para prueba                      */

/* Fijamos seed para usar la misma partición en distintos
ejemplos*/
val dataSplits = carFeaturesLabelDF.randomSplit(Array(0.66, 0.34),
seed=0)
val trainCarDF = dataSplits(0)
val testCarDF = dataSplits(1)
```

Creamos instancia `DecisionTreeClassfier` y parametrizamos

```
/* Importamos de ML     */
import org.apache.spark.ml.classification.DecisionTreeClassifier

/* Creamos una instancia de DecisionTreeClassifier                */
val Dtcar=new DecisionTreeClassifier()

/* Elegimos parámetros del modelo        */
val impureza =  "entropy"
val maxProf = 3
val maxBins =5

/* Fijamos parámetros del modelo        */
DTcar.setImpurity(impureza)
DTcar.setMaxDepth(maxProf)
DTcar.setMaxBins(maxBins)
```

Ejercicio 1: Entrenamos el clasificador, `DTcarModel_A`

- Entrenamos, examinamos y calculamos tasa de error.

Solución 1: Entrenamos el clasificador, `DTcarModel_A`

Calculamos su tasa de error

```
Tasa de error= 0,237
```

Recordar

- Con los parámetros por defecto

```
val impureza =  "entropy"
val maxProf = 5
val maxBins = 32
```

```
numNodes=23
Tasa de error= 0,190
```

Ejercicio 2: `DTcarModel_B`

- Crea árbol de decisión con

```
impureza: entropy
maxProf:10
maxBins=: 5
```

- Evaluar sobre las mismas particiones.

- Mostrar.

Solución 2: `DTcarModel_B`

Examinamos `DTcarModel_B`

Tasa de error `DTcarModel_B`

`Tasa de error= 0,085`

- Pero profundidad 10 y 211 nodos.

Ejercicio 3: `DTcarModel_C`

- Crea árbol de decisión con

```
impureza: entropy
maxProf:10
maxBins=: 150
```

- Evaluar sobre las mismas particiones.

- Mostrar.

Solución 3: `DTcarModel_C`

Examinamos `DTcarModel_C`

Tasa de error `DTcarModel_C`

`Tasa de error= 0,085`

- El mismo árbol que `DTcarModel_B`, (maxBins=5).

- El resultado era de esperar, pues no hay atributos continuos.
- Y usando `OneHotEncoder` todos los atributos toman valor 0/1.
- Sería suficiente **`maxBins=2.`**

5. OneHotEncoder y árboles de decisión

- Vamos a comparar un árbol utilizando la codificación **1 de K** frente a otro que no la utilice.

- En ambos casos utilizamos `StringIndexer` para obtener los índices a los valores de los atributos categóricos.

Ejercicio 4: Inducir un árbol con los parámetros por efecto sin usar OneHotEncoder, `DTcarModel_D`

- Codificar los atributos categóricos solo con `StringIndexer`

Solución 4: nueva columna `features`

Solución 4: examinamos nuevo *Data Frame*

`carFeaturesClaseDF`

Solución 4: creamos el árbol `DTcarModel_D`

Solución 4: examinamos el árbol

Solución 4: tasa de error sin usar OneHotEncoder

```
Tasa de error= 0,144
```

- Sin `OneHotEncoder`.
- Árbol de profundidad 5, 17 nodos y una tasa de error de 0,144.
- Comprobar que aumentando la profundidad a 10, se obtiene un árbol que tiene 125 nodos y una tasa de error de 0,031.

- Con `OneHotEncoder`
- Árbol de profundidad 5, 23 nodos y una tasa de error de 0,190.
- O árbol de profundidad 10, 211 nodos y una tasa de error de 0,085.

- En este caso, la opción con `StringIndexer` es preferible.
- El resultado depende del conjunto de datos y de la partición aleatoria.

OneHotEncoder y árboles de decisión

- No siempre es preferible la codificación **1 de K** con los árboles de decisión.
- Porque introducen nuevos atributos.
- Que permiten realizar particiones más pequeñas del conjunto de entrenamiento.
- Y seleccionar atributos que generalizan peor.

6. Referencias

- Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

- B. Zupan, M. Bohanec, I. Bratko, J. Demsar, “Machine Learning by Function Decomposition”, Proceedings of the Fourteenth International Conference Machine Learning (ICML'97), pp. 421 – 429, Nashville, Tennessee, July 1997.

- Decision trees. https://spark.apache.org/docs/3.5.6/ml-classification-regression.html#decision-trees. Último acceso: octubre 2025.
- DecisionTreeClassifiers. https://spark.apache.org/docs/3.5.6/ml-classification-regression.html#decision-tree-classifier . Último acceso: octubre 2025.
- Decision Trees - RDD-based API. https://spark.apache.org/docs/3.5.6/mllib-decision-tree.html. Último acceso: octubre 2025.