



Selección sencilla de un modelo de clasificación

Entregable Individual

Nataly Rocha

Escuela de Ingeniería Informática, Universidad de Valladolid
Técnicas Escalables de Análisis de Datos en entornos Big Data:
Clasificadores
2025 - 2026

1. Introducción

El presente trabajo forma parte del Proyecto Software: Construcción y validación de un modelo de clasificación usando la metodología CRISP-DM y Spark. Tiene como objetivo desarrollar un modelo de clasificación, aplicando técnicas de aprendizaje automático sobre el conjunto de datos Census Income (KDD). Este dataset, proveniente de encuestas de la Oficina del Censo de Estados Unidos (1994–1995), contiene información demográfica, educativa y laboral que permite estudiar y predecir patrones socioeconómicos.

En este entregable se realiza la exploración de parámetros del modelo concretamente maxDepth y maxBins, para identificar la configuración que cree el mejor modelo de clasificación usando un árbol de decisión. El proceso incluye la creación de particiones de entrenamiento y validación, la evaluación de distintas combinaciones de parámetros y la construcción del modelo final con los valores óptimos obtenidos.

2. Resumen Ejecutivo

2.1. Resumen ejecutivo del conjunto de datos

El conjunto de datos Census Income (KDD) contiene información censal ponderada obtenida de las Encuestas de Población Actual (Current Population Survey) de los años 1994 y 1995, realizadas por la Oficina de Censo de los Estados Unidos. El dataset está compuesto por 299.285 registros y 41 tributos tanto categóricos como numéricos, que incluyen variables como la edad, el sexo, nivel educativo, ocupación, estado civil, tamaño del empleador, número de horas trabajadas por semana y nacionalidad, entre otras.

2.2. Origen de los datos

UCI Machine Learning Repository – Census Income (KDD) Data Set <https://archive.ics.uci.edu/dataset/117/census+income+kdd>

2.3. Propósito y uso de los datos.

El objetivo principal de este conjunto de datos es predecir si el ingreso anual de una persona supera los 50.000 dólares, a partir del análisis de variables demográficas, educativas, familiares y laborales. El problema se formula como una clasificación binaria, donde la variable objetivo refleja el nivel de ingresos de cada individuo.

2.4. Tamaño del conjunto de datos.

El dataset se divide en dos subconjuntos. Data (entrenamiento): 199.523 instancias y Test (prueba): 99.762 instancias. En total, cuenta con 299.285 registros, cada uno correspondiente a una persona.

2.5. Número total y tipo de atributos.

El conjunto incluye 41 atributos, aunque uno de ellos (“instance weight”) se utiliza únicamente como peso de muestra y debe ser ignorado para el entrenamiento de clasificadores. Por tanto, el análisis se realiza sobre 40 atributos distribuidos en: 6 atributos continuos (numéricos) y 34 atributos nominales (categóricos). Las variables describen características como edad, nivel educativo, ocupación, estado civil, relación familiar, nacionalidad, horas trabajadas por semana, entre otras.

2.6. Descripción de la Clase.

La clase, denominada “PTOTVAL” (total person income), representa el rango de ingresos totales de cada individuo. Es una variable binaria con las siguientes categorías: “-50000”: ingresos menores o iguales a 50.000 USD (93,8%); “+50000”: ingresos superiores a 50.000 USD (6,2%). Esta distribución altamente desbalanceada plantea un desafío adicional para los modelos de clasificación, que deberán ser evaluados teniendo en cuenta este desequilibrio de clases.

2.7. Selección del mejor modelo

Para esta exploración se emplearon exclusivamente árboles de decisión, evaluando diversas combinaciones de los parámetros *maxDepth* y *maxBins* con el objetivo de identificar la configuración que genera el mejor desempeño del modelo. Dado el desbalance existente entre clases, se consideró fundamental analizar no solo la métrica de *accuracy*, sino también el *AUC-PR*, la matriz de confusión y el *recall* por clase, a fin de evaluar adecuadamente la capacidad del modelo para predecir la clase minoritaria, que representa el mayor desafío en este contexto.

3. Proceso de exploración

3.1. Creación de sub conjuntos de entrenamiento y validación

Para evitar usar el mismo conjunto de entrenamiento para entrenar y validar el modelo, se realizó una división del conjunto de entrenamiento. Resultando en un conjunto sub entrenamiento (*subTrain*) y otro de validación (*validation*) con una división del 75 % y 25 % respectivamente.

```
/**
 * Dividir el conjunto de entrenamiento:
 * NOTA IMPORTANTE: El conjunto de test NO se utiliza para selecc i
 de par metros
 */
def createTrainingSubsets(originalDF: DataFrame, trainingPercentage:
Double = 0.75, seed: Long): Array[Dataset[Row]] =
    originalDF.randomSplit(Array(trainingPercentage, 1 -
trainingPercentage), seed)
```

Igualmente, se agregó un seed y se verificó que los subconjuntos creados tengan una distribución parecida para evitar afectar al rendimiento del modelo. Obteniendo los siguientes resultados:

```
*** Distribución de clases en sub-train:
+-----+-----+-----+
|label| count|porcentaje|
+-----+-----+-----+
|  0.0|114941|    92.57|
|  1.0| 9232|     7.43|
+-----+-----+-----+

*** Distribución de clases en validation:
+-----+-----+-----+
|label|count|porcentaje|
+-----+-----+-----+
|  0.0|38448|    92.48|
|  1.0| 3127|     7.52|
+-----+-----+-----+
```

Figura 1: Resultados de las combinaciones de parámetros exploradas

3.2. Creación de combinaciones:

Para la elección de las combinaciones se tomó en cuenta lo siguiente:

maxDepth controla la profundidad del árbol de decisión. A mayor profundidad, mayor complejidad tiene el árbol, esto generalmente permite que el árbol tenga un mayor performance. Pero también es muy probable que a mayor profundidad el modelo se ajuste en exceso al conjunto de datos [Dua et al., 2017].

El desbalance de la clase es importante a tomar en cuenta, porque valores poco profundos puedan ignorar la clase minoritaria. Se decidió empezar con el valor por defecto y continuar aumentando hasta 15 para evitar overfit.

maxBins es el número máximo de particiones que el árbol de decisión usa para discretizar cada atributo y determinar puntos de división óptimos en cada nodo del árbol. Como su mínimo es el número máximo de valores de un atributo categórico empezamos con 52 debido al atributo ADTIND.

La razón de considerar combinaciones desde 52 hasta 100 se debe a que al igual que con la profundidad del árbol, un mayor número de compartimentos debería permitir que el modelo

se vuelva más complejo y podría ayudar al rendimiento con dimensiones de características más grandes. Pero después de un cierto punto, es poco probable que ayude más y, de hecho, podría obstaculizar el rendimiento en el conjunto de pruebas debido al overfit. [Dua et al., 2017]

Se exploraron 12 combinaciones (ver figura 2) generadas con el siguiente código:

```
val maxDepthValues = Array(5, 10, 15)
val maxBinsValues = Array(53, 60, 80, 100)
val combinations = maxDepthValues.flatMap(maxDepth => maxBinsValues.map
(maxBins => (maxDepth, maxBins)))
```

```
** Combinaciones a explorar:
maxDepth: 5 - maxBins: 53
maxDepth: 5 - maxBins: 60
maxDepth: 5 - maxBins: 80
maxDepth: 5 - maxBins: 100
maxDepth: 10 - maxBins: 53
maxDepth: 10 - maxBins: 60
maxDepth: 10 - maxBins: 80
maxDepth: 10 - maxBins: 100
maxDepth: 15 - maxBins: 53
maxDepth: 15 - maxBins: 60
maxDepth: 15 - maxBins: 80
maxDepth: 15 - maxBins: 100
```

Figura 2: Resultados de las combinaciones de parámetros exploradas

3.3. Observación importante sobre clase positiva y negativa

Las métricas empleadas evalúan la capacidad del modelo para distinguir la clase positiva (**1.0**) de la negativa (**0.0**). Tanto `BinaryClassificationEvaluator` como `MulticlassMetrics` asumen que la clase con valor más alto (**1.0**) es la positiva y la utilizan para calcular el AUC, la matriz de confusión, el recall, entre otros.

Esta distinción es relevante, ya que si la indexación de clases se define de forma incorrecta, las métricas podrían no reflejar el desempeño real del modelo sobre la clase minoritaria. En este caso, debido al desbalance, se ha realizado el cambio en la etapa de transformación para que la clase minoritaria (**-50000**) se etiqueta como **1.0** y la mayoritaria (**+50000**) como **0.0**, asegurando que las métricas se enfoquen en la clase minoritaria.

3.4. Entrenamiento y evaluación de modelos

Para evaluar el rendimiento de los árboles de decisión según las combinaciones de parámetros, se utilizaron las métricas de **MulticlassMetrics** y `BinaryClassificationMetrics`.

Dado el desbalance de clases, `BinaryClassificationMetrics` permitió analizar el desempeño mediante las áreas AUC-ROC y especialmente AUC-PR, más relevante por reflejar el equilibrio entre precisión y exhaustividad, y evidenciar si el modelo realmente identifica la clase minoritaria (**-50000**) o simplemente predice la mayoritaria.

Complementariamente, `MulticlassMetrics` proporcionó la matriz de confusión y las métricas de F1, recall y precision por clase. No se utilizaron las métricas ponderadas, ya que, debido al desbalance de clases, podrían resultar engañosas y dificultan el análisis del mejor modelo.

TODO: Revisar estas evaluaciones

```

1      def evaluateMulticlassMetrics(predictions: DataFrame):
2      MulticlassMetrics = {
3          val predictionAndLabels = predictions.select("prediction", "label")
4      .rdd.map(row =>
5          (row.getDouble(0), row.getDouble(1))
6          )
7          new MulticlassMetrics(predictionAndLabels)
8      }

```

TODO: AGREGAR LA EVALUACION DE BINARY

3.5. Resultados de la evaluación

3.5.1. Conjunto de sub entrenamiento

TODO agregar tabla y grafico

3.5.2. Conjunto de validación

TODO agregar tabla y grafico

En este caso se puede visualizar lo analizado en la exploración de datos, al haber tanto desbalance en la clase, el modelo predice de manera excelente para la clase mayoritaria, pero para la minoritaria tiene un comportamiento bastante pobre en algunas combinaciones peor que aleatoreo.

TODO colocar el analisis de OVERFIT haciendo la comparacion de las tablas.

El mejor modelo seleccionado

TODO: AGREGAR TABLA

TODO: AGREGAR GRAFICO DE SER POSIBLE

4. Tiempo empleado

5. Conclusiones

Text Conclusiones

Referencias

- [Dua et al., 2017] Dua, R., Ghotra, M. S., and Pentreath, N. (2017 - 2017). *Machine learning with spark : develop intelligent machine learning systems with spark 2.x*. Packt, Birmingham, England ;, second edition. edition.