



---

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

---

Métricas de evaluación de clasificadores multiclase



1. Clasificación multiclase.
2. Métricas multiclase en ML.
3. Evaluación modelo árbol de decisión datos .car con ML.
4. Mllib: Métricas de evaluación para clasificación multiclase.
5. Ejemplo: evaluación modelo regresión logística con Mllib.
6. Referencias.



# 1. Clasificación Multiclase

---

- *MLlib* y *ML* denominan clasificación multiclase a la clasificación no binaria.
- La etiqueta de clase puede tomar mas de dos valores.
- Cada instancia **solo puede pertenecer a una** clase.

# Métricas multiclase en ML

- ML proporciona la clase `MulticlassClassificationEvaluator`.
- Es un Evaluator

- Se crea una instancia de métrica

```
import  
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator  
val mceval= new MulticlassClassificationEvaluator()
```

- Se aplica con método `evaluate`: requiere `DataFrame`(o `Dataset`, no un `RDD`) con columnas de predicciones y etiquetas  
`mceval.evaluate(PredictionsandLabelsDF)`

- Métrica por defecto: "f1"

## 2. ML MulticlassClassificationEvaluator: parámetros

- `beta`: valor que controla la ponderación precision vs. recall,
  - Se utiliza en `"weightedFMeasure"`, `"fMeasureByLabel"`.
  - $\geq 1$ . Por defecto `1.0`
- `labelCol`: columna con las clases reales
  - Tipo `String`, por defecto `"label"`
- `metricLabel`: para qué clase se calcula la métrica en `"truePositiveRateByLabel"`, `"falsePositiveRateByLabel"`, `"precisionByLabel"`, `"recallByLabel"`, `"fMeasureByLabel"`
  - $\geq 0$ . Por defecto `0.0`.

# ML MulticlassClassificationEvaluator: parámetros

- `metricName`: métrica a calcular, "f1" (default), "accuracy", "weightedPrecision", "weightedRecall", "weightedTruePositiveRate", "weightedFalsePositiveRate", "weightedFMeasure", "truePositiveRateByLabel", "falsePositiveRateByLabel", "precisionByLabel", "recallByLabel", "fMeasureByLabel", "logLoss", "hammingLoss"
  - Tipo String, por defecto "f1"
- Recordar: La tasa de acierto es "accuracy".
- `predictionCol`: columna con las clases predichas
  - Tipo String, por defecto "prediction".
- `probabilityCol`: columna con las probabilidades predichas
  - Tipo String, por defecto "probability".

### 3. Evaluación árbol de decisión car.data con ML

- <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, categóricos.
- 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.



# Archivo car.data

- 1728 instancias, 6 atributos + clase.

```
~/Datos$ vi car.data  
vhigh,vhigh,2,2,small,low,unacc  
vhigh,vhigh,2,2,small,med,unacc  
vhigh,vhigh,2,2,small,high,unacc  
vhigh,vhigh,2,2,med,low,unacc  
vhigh,vhigh,2,2,med,med,unacc  
vhigh,vhigh,2,2,med,high,unacc  
vhigh,vhigh,2,2,big,low,unacc
```

- Atributos y clase, categóricos.
- 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 %)



# Creamos el modelo y predecimos

---

```
// Recuperamos el ejemplo de los coches con árboles de
decisión
// Codificando atributos categóricos sin usar oneHotencoder

:load arbolesDecisionMLstringIndexer.scala

//
// Predecimos con el modelo entrenado y creamos
predictionsAndLabels
//
val predictionsAndLabels = DTcarModel_D.transform(testCarDF)
```

```
println(f"%nData Frame predictionsAndLabels:")
predictionsAndLabels.show(5)
```

Data Frame predictionsAndLabels:

```
+-----+-----+-----+-----+-----+
|          features|label|          rawPrediction|          probability|prediction|
+-----+-----+-----+-----+-----+
|          (6,[],[])|  1.0|[0.0,109.0,0.0,23.0]|[0.0,0.8257575757...|          1.0|
|(6,[0,1],[1.0,1.0])|  3.0|[0.0,3.0,28.0,31.0]|[0.0,0.0483870967...|          3.0|
|(6,[0,1],[2.0,2.0])|  3.0|[0.0,3.0,28.0,31.0]|[0.0,0.0483870967...|          3.0|
|(6,[0,2],[3.0,2.0])|  1.0|[0.0,109.0,0.0,23.0]|[0.0,0.8257575757...|          1.0|
|(6,[0,2],[3.0,3.0])|  1.0|[0.0,109.0,0.0,23.0]|[0.0,0.8257575757...|          1.0|
+-----+-----+-----+-----+-----+
```

only showing top 5 rows



## Calculamos tasas acierto/error

---

```
//Creamos instancia de MulticlassClassificationEvaluator
//con métrica "accuracy"

import
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
val ML_multiclasmetrics = new
MulticlassClassificationEvaluator().setMetricName("accuracy")

// Calculamos la tasa de error
//
ML_multiclasmetrics.setMetricName("accuracy")
val tasa_acierto = ML_multiclasmetrics.evaluate(predictionsAndLabels)
val tasa_error = 1.0 - tasa_acierto
```



# Tasa acierto/error

```
println("Tasas de acierto/error del clasificador con ML,
métrica multiclase")
println(f"metricName=accuracy, resto de parámetros por
defecto:%n Tasa de acierto = $tasa_acierto%1.4f%n Tasa de
error = $tasa_error%1.4f %n")
```

Tasas de acierto/error del clasificador con ML, métrica multiclase

metricName=accuracy, resto de parámetros por defecto:

Tasa de acierto = 0,8562

Tasa de error = 0,1438

# Recuperamos las etiquetas de clase

```
// Obtenemos categorías originales de las etiquetas
//
import org.apache.spark.ml.feature.IndexToString

val labelsDF= predictionsAndLabels.select("label").distinct
val converter = new
IndexToString().setInputCol("label").setOutputCol("Clase original")
val clasesDF = converter.transform(labelsDF)
println(f"%nMapping índices-etiquetas:")
clasesDF.show
```

Mapping índices-etiquetas:

```
+-----+-----+
|label|Clase original|
+-----+-----+
|  0.0|          vgood|
|  1.0|          unacc|
|  3.0|           acc|
|  2.0|          good|
+-----+-----+
```

## Calculamos tasa de ciertos positivos por clase

```
// Calculamos truePositiveRateByLabel para cada etiqueta
//
ML_multiclasmetrics.setMetricName("truePositiveRateByLabel")
val labels= Array(0.0, 1.0, 2.0 , 3.0)

println(f"%nTasa de ciertos positivos por etiqueta:")
labels.foreach {l =>
    ML_multiclasmetrics.setMetricLabel(l)
    val tp =
ML_multiclasmetrics.evaluate(predictionsAndLabels)
    println(f"truePositiveRateByLabel($l) = $tp%1.4f")}
```

# Examinamos frente a ponderada

Tasa de ciertos positivos por etiqueta:

truePositiveRateByLabel(0.0) = 0,9091

truePositiveRateByLabel(1.0) = 0,9781

truePositiveRateByLabel(2.0) = 0,0000

truePositiveRateByLabel(3.0) = 0,6581

```
// Calculamos truePositiveRate ponderada
```

```
//
```

```
ML_multiclasmetrics.setMetricName("weightedTruePositiveRate")
```

```
val ponderada
```

```
=ML_multiclasmetrics.evaluate(predictionsAndLabels)
```

```
println(f"f%nPrecision ponderada: $ponderada%1.4f")
```

Tasa de ciertos positivos ponderada: 0,8562

# Cuidado con las métricas ponderadas

- Tasa de ciertos positivos ponderada: 0.8562.
- Puede aparecer un valor razonable.

## ■ Pero:

**truePositiveRateByLabel(2.0) = 0,0000 (clase good).**

**truePositiveRateByLabel(3.0) = 0,6581 (clase acc).**

- 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 %)

## Calculamos tasa de falsos positivos por clase

```
// Calculamos falsePositiveRateByLabel para cada etiqueta
//
ML_multiclasmetrics.setMetricName("falsePositiveRateByLabel")

println(f"%nTasa de falsos positivos por etiqueta:")
labels.foreach {l =>
    ML_multiclasmetrics.setMetricLabel(l)
    val fp =
ML_multiclasmetrics.evaluate(predictionsAndLabels)
    println(f"falsePositiveRateByLabel($l) = $fp%1.4f")}
```

# Examinamos frente a ponderada

Tasa de falsos positivos por etiqueta:

falsePositiveRateByLabel(0.0) = 0,0322

falsePositiveRateByLabel(1.0) = 0,2438

falsePositiveRateByLabel(2.0) = 0,0000

falsePositiveRateByLabel(3.0) = 0,0438

```
// Calculamos falsePositiveRate ponderada
```

```
//
```

```
ML_multiclasmetrics.setMetricName("weightedFalsePositiveRate")
```

```
val ponderada =ML_multiclasmetrics.evaluate(predictionsAndLabels)
```

```
println(f"%nTasa de falsos positivos ponderada: $ponderada%1.4f")
```

Tasa de falsos positivos ponderada: 0,1760

## De forma similar, para f-measure

f-measure por etiqueta:

FMeasureByLabel(0.0) = 0,6557

FMeasureByLabel(1.0) = 0,9327

FMeasureByLabel(2.0) = 0,0000

FMeasureByLabel(3.0) = 0,7365

Fmeasure ponderada: 0,8365

Métrica por defecto, f1: 0,8365



# Cuidado con las métricas ponderadas

---

- Oculta la estructura de la métrica.
- Se pondera por número de instancias por clase.
- Especialmente en clases con pocas instancias: influyen muy poco en el valor ponderado.



## 4. MLlib: Métricas de evaluación para clasificación multiclase

---

- MLlib proporciona la clase `MulticlassMetrics`.
- NO confundir con `MultilabelMetrics`.
- En el paquete  
`org.apache.spark.mllib.evaluation.MulticlassMetrics`.




# MLbib MulticlassMetrics: argumentos

- Creamos una nueva instancia **para cada** conjunto de predicciones a evaluar.

```
import org.apache.spark.mllib.evaluation.MulticlassMetrics
val metrics = new MulticlassMetrics(predictionAndLabels)
```

- Con `predictionAndLabels` un RDD con tuplas de
  - (prediction, label, weight, probability) .
  - (prediction, label, weight) .
  - (prediction, label) .

# MLlib MulticlassMetrics: métodos

- Creamos una nueva instancia **para cada** conjunto de predicciones a evaluar
  - accuracy
  - **confusionMatrix** 
  - fMeasure(label)
  - truePositiveRate(label)
  - falsePositiveRate(label)
  - precision(label)
  - recall(label)
  - weightedPrecision
  - weightedRecall
  - weightedTruePositiveRate
  - weightedFalsePositiveRate
  - weightedFMeasure
  - logLoss
  - hammingLoss

Solo en MLib  
MulticlassMetrics

## 5. Ejemplo: evaluación modelo regresión logística con MLlib

- Utilizaremos el conjunto de datos “sample\_multiclass\_classification\_data.txt”, disponible en [https://github.com/apache/spark/blob/master/data/mllib/sample\\_multiclass\\_classification\\_data.txt](https://github.com/apache/spark/blob/master/data/mllib/sample_multiclass_classification_data.txt)
- Este conjunto de datos se ha elaborado a partir del conjunto de datos Iris, y tiene 3 etiquetas de clase.
- Examinamos el ejemplo proporcionado en: <https://spark.apache.org/docs/3.5.6/mllib-evaluation-metrics.html> (solo se ha modificado para presentar los resultados con 4 decimales).

## Leemos los datos y creamos conjuntos training y test

```
import org.apache.spark.mllib.util.MLUtils
// donde estén los datos
val PATH = "/home/usuario/Datos/"
val file = "sample_multiclass_classification_data.txt"

val data = MLUtils.loadLibSVMFile(sc, PATH + file)

// Split data into training (60%) and test (40%)
val Array(training, test) =
  data.randomSplit(Array(0.6, 0.4), seed = 11L)

training.cache()
```



# Creamos el modelo

---

```
import  
org.apache.spark.mllib.classification.LogisticRegressionWithLBFGS  
  
// Run training algorithm to build the model  
val model = new LogisticRegressionWithLBFGS()  
    .setNumClasses(3).run(training)
```

# Realizamos predicciones con el modelo

```
import org.apache.spark.mllib.regression.LabeledPoint
// Compute raw scores on the test set
val predictionAndLabels = test.map { case
  LabeledPoint(label, features) => val prediction =
    model.predict(features)
    (prediction, label) }
//No parecen raw scores
println(f"%nEjemplo modelo Logistic Regresion, MLlib,
  métrica multiclase.%npredictionAndLabels RDD")
predictionAndLabels.take(5).foreach(x => println(x))
Ejemplo modelo Logistic Regresion, MLlib, métrica multiclase.
predictionAndLabels RDD
(1.0,1.0)
(2.0,2.0)
(0.0,0.0)
(1.0,1.0)
(1.0,1.0)
```

# Nueva instancia de métrica y matriz de confusión

```
import
org.apache.spark.mllib.evaluation.MulticlassMetrics

// Instantiate metrics object
val metrics = new
MulticlassMetrics(predictionAndLabels)

// Confusion matrix
println("Confusion matrix:")
println(metrics.confusionMatrix)
```

```
15.0  0.0  2.0
0.0   17.0 0.0
5.0   0.0  16.0
```

```
// Overall Statistics  
val accuracy = metrics.accuracy  
println("Summary Statistics")  
println(f"Accuracy = $accuracy%1.4f")
```

Summary Statistics

Accuracy = 0.8727

## *precision, recall* (por clase)

```
// Precision by label
val labels = metrics.labels
labels.foreach {l => val pl = metrics.precision(l)
    println(f"Precision($l) = $pl%1.4f")}
```

```
Precision(0.0) = 0.7500
```

```
Precision(1.0) = 1.0000
```

```
Precision(2.0) = 0.8889
```

```
// Recall by label
labels.foreach {l => val rl = metrics.recall(l)
    println(f"Recall($l) = $rl%1.4f")}
```

```
Recall(0.0) = 0.8824
```

```
Recall(1.0) = 1.0000
```

```
Recall(2.0) = 0.7619
```

## Tasa de fp y F-measure (por clase)

```
// False positive rate by label
labels.foreach {l => val fpl = metrics.falsePositiveRate(l)
    println(f"falsePositiveRate($l) = $fpl%1.4f")}
falsePositiveRate(0.0) = 0.1316
falsePositiveRate(1.0) = 0.0000
falsePositiveRate(2.0) = 0.0588

// F-measure by label
labels.foreach {l => val f1l = metrics.fMeasure(l)
    println(f"F1-score($l) = $f1l%1.4f")}
F1-score(0.0) = 0.8108
F1-score(1.0) = 1.0000
F1-score(2.0) = 0.8205
```

```
// Weighted stats  
println(f"Weighted precision: ${metrics.weightedPrecision}%1.4f")
```

Weighted precision: 0.8803

```
println(f"Weighted recall: ${metrics.weightedRecall}%1.4f")
```

Weighted recall:0.8727

```
println(f"Weighted F1 score: ${metrics.weightedFMeasure}%1.4f")
```

Weighted F1 score: 0,8730

```
println(f"Weighted false positive rate:  
${metrics.weightedFalsePositiveRate}%1.4f")
```

Weighted false positive rate: 0.0631

## 6. Referencias

- MulticlassClassificationEvaluator.  
<https://spark.apache.org/docs/3.5.6/api/scala/org/apache/spark/ml/evaluation/MulticlassClassificationEvaluator.html>. Último acceso: octubre 2025.
- Evaluation Metrics - RDD-based API.  
<https://spark.apache.org/docs/3.5.6/mllib-evaluation-metrics.html>. Último acceso: octubre 2025.
- MulticlassMetrics.  
<https://spark.apache.org/docs/3.5.6/api/scala/org/apache/spark/mllib/evaluation/MulticlassMetrics.html>. Último acceso: octubre 2025.
- 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.
- “sample\_multiclass\_classification\_data.txt”, disponible en [https://github.com/apache/spark/blob/master/data/mllib/sample\\_multiclass\\_classification\\_data.txt](https://github.com/apache/spark/blob/master/data/mllib/sample_multiclass_classification_data.txt). Último acceso: octubre 2025.