UNIVERSIDAD DE GRANADA

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

CLOUD COMPUTING

CURSO 2024 / 2025

Práctica 3: Spark

Trabajo realizado por:
Mario Martínez Sánchez
(DNI: 23310000Y)

Índice

1. Introducción

En esta práctica se plantea un problema de clasificación binaria utilizando Apache Spark y su librería MLlib. El objetivo es construir y evaluar distintos clasificadores aplicados a datos relacionados con objetos celestes, usando tres versiones del conjunto de datos: small_celestial.csv, medium_celestial.csv y half_celestial.csv.

A través de estos datasets, se busca predecir el tipo de objeto a partir de sus características físicas. Para ello, se utilizan tres algoritmos de clasificación ampliamente usados: Regresión Logística, Random Forest y SVM, probando con diferentes combinaciones de hiperparámetros.

Además, se aborda el problema de desbalanceo de clases, común en muchos casos reales, aplicando la técnica de submuestreo aleatorio (Random UnderSampling, RUS) para equilibrar el número de instancias por clase antes del entrenamiento. Esto permite mejorar la calidad de las predicciones y evaluar de forma justa el rendimiento de cada modelo.

2. Resolución del problema de clasificación

2.1. Análisis exploratorio

Una vez seleccionado el dataset, este se carga utilizando spark.read.csv, con la opción inferSchema=True para permitir que Spark infiera automáticamente el tipo de datos de cada columna. Tras la carga, se lleva a cabo un análisis exploratorio inicial que permite comprender mejor la estructura y el contenido de los datos. Este análisis incluye:

- **Distribución de clases**: se calcula mediante *df.groupBy("type").count()* para conocer el número de instancias por clase y detectar posibles desbalances.
- **Detección de valores nulos**: usando la expresión *count(when(col(c).isNull(), c))* para cada columna, se identifican posibles valores ausentes que podrían afectar al entrenamiento.
- **Estadísticas descriptivas:** se obtienen con *df.describe().show()* para analizar medidas como la media, desviación estándar, mínimo y máximo de cada atributo numérico.

2.2. Preprocesamiento

El proceso de preparación de los datos se compone de las siguientes etapas:

- **Indexación de etiquetas**: Se convierte la variable categórica objetivo (type) en una variable numérica mediante StringIndexer, lo que permite utilizarla en los modelos de clasificación de Spark MLlib.
- **Balanceo de clases:** Dado que el conjunto de datos presenta un desequilibrio entre clases, se aplica una técnica de submuestreo aleatorio (Random UnderSampling) para igualar el número de instancias de cada clase y evitar que los modelos favorezcan la clase mayoritaria.

- **Filtrado por clase**: Se separan las clases para aplicar el submuestreo de forma controlada y, posteriormente, se combinan las clases balanceadas en un único conjunto de datos.
- **Ensamblado, escalado y normalización:** Se utiliza un VectorAssembler para combinar las características en un único vector. Luego, los datos se escalan mediante StandardScaler y se normalizan con MinMaxScaler para mejorar el rendimiento y la estabilidad de los algoritmos de clasificación.

Para el balanceo de clases explicado anteriormente, se emplea el siguiente código:

```
## Balanceo mediante Submuestreo Aleatorio (RUS)
counts = df.groupBy("label").count().collect()
label_counts = {row['label']: row['count'] for row in counts}
min_count = min(label_counts.values())

print(f"\nBalanceando clases con submuestreo al tamaño de la clase
minoritaria = {min_count}")
```

2.3. Partición y evaluación del dataset

El conjunto se divide en 80% entrenamiento y 20% prueba y se entrenan tres modelos, cada uno con dos configuraciones distintas de hiperparámetros. Estos modelos son Logistic Regression, Random Forest y SVM.

```
## Logistic Regression
lr1        =        LogisticRegression(featuresCol="normalizedFeatures",
labelCol="label", maxIter=10, regParam=0.3)
lr2        =        LogisticRegression(featuresCol="normalizedFeatures",
labelCol="label", maxIter=20, regParam=0.1)

lr_model1 = lr1.fit(train)
lr_model2 = lr2.fit(train)

auc_lr1 = evaluator.evaluate(lr_model1.transform(test))
auc_lr2 = evaluator.evaluate(lr_model2.transform(test))

## Random Forest
rf1      =      RandomForestClassifier(featuresCol="normalizedFeatures",
labelCol="label", numTrees=10, maxDepth=5)
rf2      =      RandomForestClassifier(featuresCol="normalizedFeatures",
labelCol="label", numTrees=50, maxDepth=10)

rf_model1 = rf1.fit(train)
rf_model2 = rf2.fit(train)
```

```
auc_rf1 = evaluator.evaluate(rf_model1.transform(test))
auc_rf2 = evaluator.evaluate(rf_model2.transform(test))

## SVM
svm1  =  LinearSVC(featuresCol="normalizedFeatures",  labelCol="label",
maxIter=10, regParam=0.1)
svm2  =  LinearSVC(featuresCol="normalizedFeatures",  labelCol="label",
maxIter=20, regParam=0.01)

svm_model1 = svm1.fit(train)
svm_model2 = svm2.fit(train)

auc_svm1 = evaluator.evaluate(svm_model1.transform(test))
auc_svm2 = evaluator.evaluate(svm_model2.transform(test))
```

3. Modo de ejecución

La ejecución del proyecto se realiza en un entorno basado en contenedores, utilizando Docker y Docker Compose para facilitar la configuración y despliegue de Spark. Los pasos para ejecutar la práctica son los siguientes:

- **Inicialización del entorno:** se levanta el entorno definido en el archivo docker-compose.yml con el siguiente comando:

  docker compose up

- **Acceso al contenedor de Spark**: una vez iniciado el entorno, se accede al contenedor que ejecuta Spark con:

  docker exec -it spark bash

- **Ejecución en función del conjunto de prueba**: dentro del contenedor, se ejecuta el script code.py pasando como argumento el identificador del dataset que se desea utilizar:
  1. Para el dataset small_celestial.csv:

     spark-submit /scripts/code.py 0

  2. Para el dataset medium_celestial.csv:

     spark-submit /scripts/code.py 1

  3. Para el dataset half_celestial.csv (de mayor tamaño), se recomienda ajustar los recursos asignados al driver y al executor para evitar errores de memoria:

spark-submit --executor-memory 4G --driver-memory 4G /scripts/code.py 2

Estos comandos lanzan la ejecución del pipeline de clasificación definido en el script, incluyendo el preprocesamiento, balanceo de clases, entrenamiento de modelos y evaluación final.

4. Resultados obtenidos

Para mostrar los resultados obtenidos con estos dataset, se mostrarán a continuación tres tablas, una para cada conjunto de datos utilizado.

4.1. small_celestial.csv

Clasificador	Parámetros	AUC ROC
Logistic Regression	maxIter=10, regParam=0.3	0.815
	maxIter=20, regParam=0.1	0.835
Random Forest	numTrees=10, maxDepth=5	0.906
	numTrees=50, maxDepth=10	0.967
Linear SVM	maxIter=10, regParam=0.1	0.845
	maxIter=20, regParam=0.01	0.891

4.2. medium_celestial.csv

Clasificador	Parámetros	AUC ROC
Logistic Regression	maxIter=10, regParam=0.3	0.816
	maxIter=20, regParam=0.1	0.836
Random Forest	numTrees=10, maxDepth=5	0.903
	numTrees=50, maxDepth=10	0.968
Linear SVM	maxIter=10, regParam=0.1	0.845
	maxIter=20, regParam=0.01	0.891

4.3. half_celestial.csv

Clasificador	Parámetros	AUC ROC
Logistic Regression	maxIter=10, regParam=0.3	0.817
	maxIter=20, regParam=0.1	0.837
Random Forest	numTrees=10, maxDepth=5	0.907
	numTrees=50, maxDepth=10	0.968
Linear SVM	maxIter=10, regParam=0.1	0.646
	maxIter=20, regParam=0.01	0.89

5. Conclusiones

Tras la ejecución de los experimentos sobre los tres conjuntos de datos (small_celestial.csv, medium_celestial.csv y half_celestial.csv), se puede observar una tendencia clara en el rendimiento de los diferentes algoritmos de clasificación utilizados. La métrica empleada para la evaluación ha sido el AUC ROC, que permite comparar el desempeño general de los modelos independientemente del umbral de decisión.

Los resultados muestran que:

- **Random Forest** es el algoritmo que consistentemente obtiene los mejores resultados en los tres datasets, especialmente cuando se incrementan sus hiperparámetros (numTrees=50, maxDepth=10). En todos los casos, esta configuración supera el 0.96 de AUC, destacando su robustez y capacidad para modelar datos complejos.
- **Logistic Regression** ofrece un rendimiento sólido, con valores de AUC en torno al 0.83, ligeramente superiores en la configuración con más iteraciones y menor regularización (maxIter=20, regParam=0.1). Aunque no alcanza los valores de Random Forest, su simplicidad y eficiencia pueden hacerla adecuada en contextos con recursos limitados o necesidad de interpretabilidad.
- **Linear SVM** presenta resultados mixtos. En los datasets más pequeños (small y medium), su rendimiento es razonablemente alto (hasta 0.891), pero en el dataset más grande (half_celestial.csv), su desempeño cae significativamente en la configuración más simple (AUC = 0.646), lo que indica una mayor sensibilidad a la escala del problema y a la elección de parámetros.

En resumen, Random Forest con mayor número de árboles y profundidad es la opción más eficaz para este problema de clasificación binaria. No solo obtiene el mayor AUC en todos los casos, sino que también mantiene su rendimiento de forma estable a medida que crece el tamaño del dataset. Esto sugiere que es el modelo más adecuado para escenarios similares donde la precisión es prioritaria.