SESIÓN PROCESAMIENTO DE DATOS

CONTENIDOS:

- Qué es MLlib.
- Características de MLlib y MLlib Tools.
- Estructuras de dato de MLlib.
- Algoritmos de Machine Learning soportados por MLlib.
- Implementación de algoritmos de Machine Learning supervisados con MLlib.
- Implementación de algoritmos de Machine Learning no supervisados con MLlib.

QUÉ ES MLLIB

MLlib es la biblioteca de Machine Learning de **Apache Spark**, diseñada para proporcionar algoritmos y utilidades optimizadas para el procesamiento distribuido. Su objetivo es facilitar la implementación de modelos de aprendizaje automático en entornos de **Big Data** mediante **Spark RDDs (*Resilient Distributed Datasets*)** y **DataFrames**.

MLlib se usa principalmente para realizar tareas como clasificación, regresión, clustering, reducción de dimensionalidad y filtrado colaborativo, aprovechando la potencia de Spark para manejar grandes volúmenes de datos.

CARACTERÍSTICAS DE MLLIB Y MLLIB TOOLS

Características principales de MLlib:

1. **Escalabilidad:** Optimizado para entornos distribuidos, permitiendo el procesamiento de grandes volúmenes de datos.
2. **Alto rendimiento:** Utiliza computación distribuida sobre Apache Spark, lo que permite un rendimiento superior en comparación con otras bibliotecas.
3. **Fácil integración:** Compatible con APIs de **Python (PySpark), Scala, Java y R**.
4. **Conjunto de algoritmos optimizados:** Incluye algoritmos de aprendizaje supervisado y no supervisado, además de herramientas para **preprocesamiento y extracción de características**.

5. **Soporte para modelos en DataFrames y RDDs:** Permite trabajar con estructuras de datos eficientes y escalables.

MLlib Tools (Herramientas complementarias)

MLlib también proporciona herramientas adicionales para mejorar el flujo de trabajo del Machine Learning:

- **ML Pipelines:** Permite estructurar modelos de ML en una secuencia ordenada de transformación y ajuste.
- **ML Tuning (Hyperparameter Tuning):** Métodos como *Cross-Validation* y *Grid Search* para optimizar hiperparámetros.
- **Persistencia de Modelos:** Permite guardar y cargar modelos entrenados para su reutilización.

ESTRUCTURAS DE DATO DE MLLIB

MLlib utiliza **dos tipos principales de estructuras de datos** para representar y procesar datos de Machine Learning en Spark:

1. **RDDs (Resilient Distributed Datasets):**
    - Estructura de datos distribuida e inmutable que permite operaciones paralelas.
    - Se usa en versiones anteriores de MLlib, pero sigue siendo útil para ciertos escenarios.
2. **DataFrames:**
    - Forma más optimizada y eficiente de manejar datos en Spark.
    - Ofrece mayor facilidad para manipular y transformar datos con funciones SQL y optimizaciones automáticas.
    - Se recomienda usar DataFrames en lugar de RDDs para mayor rendimiento y compatibilidad con Spark ML.

ALGORITMOS DE MACHINE LEARNING SOPORTADOS POR MLLIB

MLlib ofrece una amplia variedad de algoritmos de aprendizaje automático. Se dividen en dos grandes grupos:

1. Algoritmos Supervisados:

Estos algoritmos aprenden a partir de datos etiquetados para hacer predicciones sobre nuevos datos. Algunos ejemplos en MLlib incluyen:

- **Regresión Lineal** y **Regresión Logística**
- **Árboles de Decisión** y **Random Forest**
- **Gradient-Boosted Trees (GBTs)**
- **Support Vector Machines (SVMs)**
- **Naïve Bayes**

2. Algoritmos No Supervisados:

Se utilizan cuando no se tienen etiquetas y el objetivo es encontrar patrones en los datos. Algunos algoritmos en MLlib incluyen:

- **Clustering con K-Means**
- **Latent Dirichlet Allocation (LDA) para modelado de temas**
- **Algoritmos de reducción de dimensionalidad como PCA (Principal Component Analysis)**
- **Filtrado Colaborativo para sistemas de recomendación**

IMPLEMENTACIÓN DE ALGORITMOS DE MACHINE LEARNING SUPERVISADOS CON MLLIB

Para entrenar modelos supervisados con MLlib, se siguen estos pasos generales:

1. **Carga de datos:** Se leen los datos en un DataFrame de Spark.
2. **Preprocesamiento:** Se convierten los datos en el formato adecuado, generalmente con *VectorAssembler*.
3. **División del dataset:** Se divide en **train** y **test**.
4. **Entrenamiento del modelo:** Se ajusta el modelo con el conjunto de entrenamiento.

5. **Evaluación:** Se mide el desempeño en el conjunto de prueba con métricas apropiadas.
6. **Predicción:** Se usan nuevos datos para obtener predicciones.

Ejemplo: Regresión Logística en PySpark MLlib

```python
from pyspark.sql import SparkSession
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler

# Crear sesión de Spark
spark = SparkSession.builder.appName("MLlib_Supervised").getOrCreate()

# Cargar datos en un DataFrame
data = spark.read.csv("datos.csv", header=True, inferSchema=True)

# Transformar características en un solo vector
assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")
data = assembler.transform(data)

# Dividir en train y test
train, test = data.randomSplit([0.8, 0.2])

# Crear modelo de regresión logística
lr = LogisticRegression(featuresCol="features", labelCol="label")

# Entrenar modelo
model = lr.fit(train)

# Realizar predicciones
predictions = model.transform(test)

# Mostrar resultados
predictions.select("label", "prediction").show()
```

Ilustración 1 Ejemplo regresión logistica

IMPLEMENTACIÓN DE ALGORITMOS DE MACHINE LEARNING NO SUPERVISADOS CON MLLIB

Los algoritmos no supervisados siguen un flujo similar, pero sin una variable objetivo. Se centran en **agrupar datos o reducir dimensiones**.

Ejemplo: Clustering con K-Means en PySpark MLlib

```python
from pyspark.ml.clustering import KMeans

# Crear modelo K-Means con 3 clusters
kmeans = KMeans(featuresCol="features", k=3)

# Entrenar modelo
model = kmeans.fit(train)

# Realizar predicciones de clusters
clusters = model.transform(test)

# Mostrar resultados
clusters.select("features", "prediction").show()
```

Ilustración 2 Clustering con K-Means

ACTIVIDAD PRÁCTICA GUIADA: Clasificación de flores Iris con MLlib en PySpark

Objetivo: Entrenar un modelo de Regresión Logística con MLlib en PySpark utilizando el **dataset Iris**, aplicando preprocesamiento, entrenamiento y evaluación del modelo.

Dataset:

Usaremos el dataset *Iris*, que contiene 150 registros con las siguientes columnas:

- sepal_length (longitud del sépalo en cm)
- sepal_width (ancho del sépalo en cm)
- petal_length (longitud del pétalo en cm)
- petal_width (ancho del pétalo en cm)
- species (tipo de flor: Setosa, Versicolor, Virginica)

Conversión de datos: Como MLlib requiere valores numéricos para la clasificación, convertiremos species en valores numéricos:

- Setosa → 0
- Versicolor → 1
- Virginica → 2

Paso 1: Configurar el entorno y cargar los datos

```python
from pyspark.sql import SparkSession

# Crear sesión de Spark
spark = SparkSession.builder.appName("MLlib_Iris_Clasificacion").getOrCreate()

# Cargar datos en un DataFrame desde un archivo CSV
data = spark.read.csv("iris.csv", header=True, inferSchema=True)

# Mostrar primeras filas del dataset
data.show(5)
```

Paso 2: Preprocesamiento de datos

Conversión de la columna species a valores numéricos

```python
from pyspark.sql.functions import when, col

# Convertir species en valores numéricos
data = data.withColumn("label", when(col("species") == "setosa", 0)
                                .when(col("species") == "versicolor", 1)
                                .when(col("species") == "virginica", 2))

# Mostrar la transformación
data.select("species", "label").distinct().show()
```

Convertir características en un solo vector para MLlib

```python
from pyspark.ml.feature import VectorAssembler

# Definir las columnas de características
feature_cols = ["sepal_length", "sepal_width", "petal_length", "petal_width"]

# Unir las características en un solo vector
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
data = assembler.transform(data)

# Seleccionar solo columnas necesarias
data = data.select("features", "label")

# Mostrar la transformación
data.show(5)
```

Paso 3: División en entrenamiento y prueba

```
# Dividir los datos en conjunto de entrenamiento (80%) y prueba (20%)
train, test = data.randomSplit([0.8, 0.2], seed=42)
```

Paso 4: Entrenar el modelo de Regresión Logística

```
from pyspark.ml.classification import LogisticRegression

# Definir el modelo de Regresión Logística
lr = LogisticRegression(featuresCol="features", labelCol="label", maxIter=10)

# Entrenar el modelo con los datos de entrenamiento
model = lr.fit(train)
```

Paso 5: Realizar predicciones

```
# Aplicar el modelo al conjunto de prueba
predictions = model.transform(test)

# Mostrar predicciones
predictions.select("label", "prediction").show(10)
```

Paso 6: Evaluar el modelo

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Evaluar la precisión del modelo
evaluator = MulticlassClassificationEvaluator(labelCol="label", metricName="accuracy")

# Calcular precisión
accuracy = evaluator.evaluate(predictions)
print(f"Precisión del modelo: {accuracy:.2f}")
```