Question: Write Spark MLlib code for Decision Tree Classification for MNIST with CrossValidator autotuner.

Submitted by Shramana Sinha, 23f1002703

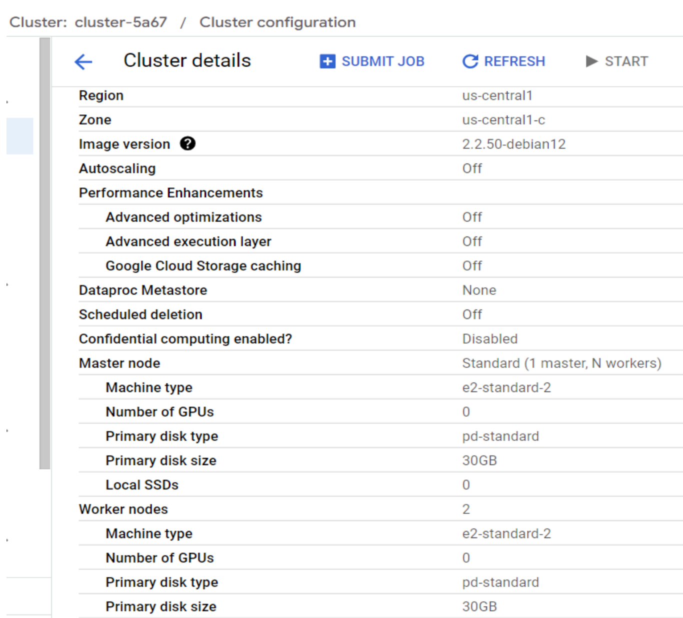
1. Cluster Setup

A Google Cloud Dataproc cluster was created with the following configuration:

- 1 master node (e2-standard-2)
- 2 worker nodes (e2-standard-2)
- 30GB boot disk for each node
- Debian 12 with Dataproc image version 2.2
- Jupyter component enabled
- Public IP address configured

The cluster was provisioned using the following command in the google cloud shell:

```
Unset
gcloud dataproc clusters create cluster-5a67 --enable-component-gateway
--region us-central1 --master-machine-type e2-standard-2
--master-boot-disk-size 30 --num-workers 2 --worker-machine-type e2-standard-2
--worker-boot-disk-size 30 --image-version 2.2-debian12 --optional-components
JUPYTER --project celtic-guru-448518-f8 --public-ip-address
```


Cluster: cluster-5a67 / Cluster configuration

Cluster details   SUBMIT JOB   REFRESH   START

Region	us-central1
Zone	us-central1-c
Image version	2.2.50-debian12
Autoscaling	Off
Performance Enhancements	
Advanced optimizations	Off
Advanced execution layer	Off
Google Cloud Storage caching	Off
Dataproc Metastore	None
Scheduled deletion	Off
Confidential computing enabled?	Disabled
Master node	Standard (1 master, N workers)
Machine type	e2-standard-2
Number of GPUs	0
Primary disk type	pd-standard
Primary disk size	30GB
Local SSDs	0
Worker nodes	2
Machine type	e2-standard-2
Number of GPUs	0
Primary disk type	pd-standard
Primary disk size	30GB

Screenshot: The dataproc cluster for the assignment

2. Implementation

The code from [Databricks Decision Trees Documentation]() was modified to use CrossValidator for automatic hyperparameter tuning instead of manual tuning, as required by the instructions. The modified code was executed on the Jupyter component of the cluster.

2.1 Data Acquisition and Preparation

The MNIST dataset was acquired in LibSVM format, decompressed, and loaded into HDFS for Spark processing:

```python
# Download the dataset
!wget
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/mnist.bz2
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/mnist.t.bz2

# Decompress the files
!bzip2 -d mnist.bz2 mnist.t.bz2

# Load into HDFS
!hdfs dfs -put -f mnist
!hdfs dfs -put -f mnist.t

# Read data using Spark
training = spark.read.format("libsvm").load("mnist")
test = spark.read.format("libsvm").load("mnist.t")
```

```
DataFrame[label: double, features: vector]

+-----+--------------------+
|label|            features|
+-----+--------------------+
|  5.0|(780,[152,153,154...|
|  0.0|(780,[127,128,129...|
|  4.0|(780,[160,161,162...|
|  1.0|(780,[158,159,160...|
|  9.0|(780,[208,209,210...|
|  2.0|(780,[155,156,157...|
|  1.0|(780,[124,125,126...|
|  3.0|(780,[151,152,153...|
|  1.0|(780,[152,153,154...|
|  4.0|(780,[134,135,161...|
+-----+--------------------+
only showing top 10 rows
```

Screenshot: The training data, consisting of labels and features columns

2.2 Machine Learning Pipeline Construction

A two-stage pipeline was constructed for the classification task, which consisted of:

- A `StringIndexer` to convert the label column to a format suitable for classification
- A `DecisionTreeClassifier` as the model

```python
indexer = StringIndexer(inputCol="label", outputCol="indexedLabel")
dtc = DecisionTreeClassifier(labelCol="indexedLabel")
pipeline = Pipeline(stages=[indexer, dtc])
```

3.3 Hyperparameter Tuning with CrossValidator

The weightedPrecision metric was selected as the evaluation criterion. The parameter grid encompassed 40 distinct model configurations by exploring - 8 different tree depths (0-7) and 5 different bin sizes (2, 4, 8, 16, 32). These combinations were systematically evaluated using 3-fold cross-validation.

```python
# Define evaluation metric
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel",
    predictionCol="prediction",
    metricName="weightedPrecision"
)

# Build parameter grid
paramGrid = ParamGridBuilder() \
    .addGrid(dtc.maxDepth, range(0, 8)) \
    .addGrid(dtc.maxBins, [2, 4, 8, 16, 32]) \
    .build()

# Configure cross-validation
cv = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator,
    numFolds=3
)

# Train the model
print("Training models with cross-validation...")
cvModel = cv.fit(training)
```

2.4 Model Evaluation

The best model's parameters were extracted and its performance was evaluated on both training and test datasets:

```python
# Extract best model and parameters
bestModel = cvModel.bestModel
bestPipelineModel = bestModel
bestTreeModel = bestPipelineModel.stages[-1]

# Print the best model parameters
print("Best model parameters:")
print(f"maxDepth: {bestTreeModel.getMaxDepth()}")
print(f"maxBins: {bestTreeModel.getMaxBins()}")

# Evaluate on training data
predictions_train = bestPipelineModel.transform(training)
weighted_precision_train = evaluator.evaluate(predictions_train)
print(f"Training weighted precision: {weighted_precision_train}")

# Evaluate on test data
predictions = bestPipelineModel.transform(test)
weighted_precision_test = evaluator.evaluate(predictions)
print(f"Test weighted precision: {weighted_precision_test}")
```

3. Results

The best performing model, as identified by the CrossValidator had the following configuration:

- maxDepth: 7
- maxBins: 8

```
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_4d05f4924530, depth=7, numNodes=245, numClasses=10, numFeatures=780

Best model parameters:
maxDepth: 7
maxBins: 8
```

Screenshot: The best parameters selected by the CrossValidator

This model achieved:

- Training weighted precision: 0.7915121381227309

```
Training weighted precision: 0.7915121381227309
```

Screenshot: The training weighted precision

- Test weighted precision: 0.7946790596293032

```
Test weighted precision: 0.7946790596293032
```

Screenshot: The test weighted precision