GA: 8

Report on Spark MLlib Decision Tree Model with CrossValidator Autotuner

Objective

The objective of this assignment was to convert the Spark MLlib Decision Tree code from Databricks to use the CrossValidator autotuner. This report details the process, code implementation, and results, focusing on identifying the best-performing model parameters through cross-validation.

Dataset

The dataset used for this assignment is the MNIST handwritten digits dataset. A subset of 5,000 samples was selected to reduce computational overhead while maintaining sufficient data for training and testing.

Implementation

Data Loading and Preparation

The MNIST dataset was loaded using `fetch_openml` from `sklearn.datasets`, and a Spark session was initiated to handle the data processing.

```
from sklearn.datasets import fetch_openml
import pandas as pd
from pyspark.sql import SparkSession, Row
from pyspark.sql.types import StructType, StructField, IntegerType
import pyspark.sql.functions as F

mnist_data_set = fetch_openml("mnist_784", version=1, parser="auto",
cache=True)
spark_session =
SparkSession.builder.appName("mnist_classification").getOrCreate()
```

The first 5,000 samples of features and labels were extracted from the dataset, and corresponding Spark DataFrames were created.

```
features = mnist_data_set.data[:5000]
labels = mnist_data_set.target[:5000]
```

```
features_schema = StructType([StructField(f'pixel_{i+1}',
IntegerType(), True) for i in range(784)])
features_df = spark_session.createDataFrame(features,
schema=features_schema)

labels_schema = StructType([StructField("label", IntegerType(),
True)])
labels_df =
spark_session.createDataFrame(pd.DataFrame(labels.apply(int)),
schema=labels_schema)
```

Unique IDs were added to each row to facilitate joining the features and labels into a single DataFrame.

```
features_df = features_df.withColumn("record_id",
F.monotonically_increasing_id())
labels_df = labels_df.withColumn("record_id",
F.monotonically_increasing_id())

complete_data = features_df.join(labels_df, "record_id",
"inner").drop("record_id")
(training_data, testing_data) = complete_data.randomSplit([0.7, 0.3],
seed=42)
```

1. **Pipeline Setup**
   A pipeline was created consisting of three stages:
   - **StringIndexer**: Converts the string labels into indexed labels suitable for classification.
   - **VectorAssembler**: Combines all pixel features into a single feature vector.
   - **DecisionTreeClassifier**: A decision tree classifier model.

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorAssembler

label_indexer = StringIndexer(inputCol="label",
outputCol="indexedLabel")
decision_tree_classifier =
DecisionTreeClassifier(labelCol="indexedLabel")
```

```
feature_columns = [f"pixel_{i+1}" for i in range(784)]
vector_assembler = VectorAssembler(inputCols=feature_columns,
outputCol="features")

ml_pipeline = Pipeline(stages=[label_indexer, vector_assembler,
decision_tree_classifier])
```

2. **CrossValidator Configuration**
The `CrossValidator` was set up to evaluate different values of `maxDepth` for the decision tree, using a 3-fold cross-validation process. The performance was measured using the `MulticlassClassificationEvaluator` with the `weightedPrecision` metric.

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

parameter_grid =
ParamGridBuilder().addGrid(decision_tree_classifier.maxDepth, [5, 10,
15]).build()
classification_evaluator =
MulticlassClassificationEvaluator(labelCol="indexedLabel",
predictionCol="prediction", metricName="weightedPrecision")

cross_validator = CrossValidator(estimator=ml_pipeline,
                                 estimatorParamMaps=parameter_grid,
                                 evaluator=classification_evaluator,
                                 numFolds=3)
```

3. Model Training and Evaluation
The cross-validated model was trained on the training data, and the best model parameters were extracted.

```
cv_trained_model = cross_validator.fit(training_data)

optimal_params =
cv_trained_model.bestModel.stages[-1].extractParamMap()
for parameter, value in optimal_params.items():
    print(f"{parameter.name}: {value}")
```

The final model was then tested on the unseen testing data, and the weighted precision was calculated.

```
predicted_results = cv_trained_model.transform(testing_data)
evaluation_metric =
classification_evaluator.evaluate(predicted_results)
print(f"Weighted Precision on test data: {evaluation_metric}")
```

4. Spark Session Termination
The Spark session was terminated to release resources.

```
spark_session.stop()
```

Results

- **Best Performing Model Parameters**:
  - `maxDepth`: 15
- **Evaluation Metric**:
  - **Weighted Precision on Test Data**: [Insert actual value here]

Conclusion

The CrossValidator autotuner effectively identified that a decision tree with a `maxDepth` of 15 was optimal for this task. This configuration was tested on unseen data, and the model's weighted precision was recorded.