Free Questions for Databricks-Machine-Learning-Associate by certsinside

Shared by Sheppard on 17-05-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

A data scientist has produced three new models for a single machine learning problem. In the past, the solution used just one model. All four models have nearly the same prediction latency, but a machine learning engineer suggests that the new solution will be less time efficient during inference.

In which situation will the machine learning engineer be correct?

Options:

A- When the new solution requires if-else logic determining which model to use to compute each prediction

B- When the new solution's models have an average latency that is larger than the size of the original model

C- When the new solution requires the use of fewer feature variables than the original model

D- When the new solution requires that each model computes a prediction for every record

E- When the new solution's models have an average size that is larger than the size of the original model

Answer:

D

Explanation:

If the new solution requires that each of the three models computes a prediction for every record, the time efficiency during inference will be reduced. This is because the inference process now involves running multiple models instead of a single model, thereby increasing the overall computation time for each record.

In scenarios where inference must be done by multiple models for each record, the latency accumulates, making the process less time efficient compared to using a single model.

Model Ensemble Techniques

Question 2

Question Type: MultipleChoice

A data scientist has developed a machine learning pipeline with a static input data set using Spark ML, but the pipeline is taking too long to process. They increase the number of workers in the cluster to get the pipeline to run more efficiently. They notice that the number of rows in the training set after reconfiguring the cluster is different from the number of rows in the training set prior to reconfiguring the cluster.

Which of the following approaches will guarantee a reproducible training and test set for each model?

Options:

A- Manually configure the cluster

B- Write out the split data sets to persistent storage

C- Set a speed in the data splitting operation

D- Manually partition the input data

Answer:

B

Explanation:

To ensure reproducible training and test sets, writing the split data sets to persistent storage is a reliable approach. This allows you to consistently load the same training and test data for each model run, regardless of cluster reconfiguration or other changes in the environment.

Correct approach:

Split the data.

Write the split data to persistent storage (e.g., HDFS, S3).

Load the data from storage for each model training session.

train_df, test_df = spark_df.randomSplit([0.8, 0.2], seed=42) train_df.write.parquet('path/to/train_df.parquet') test_df.write.parquet('path/to/test_df.parquet') # Later, load the data train_df = spark.read.parquet('path/to/train_df.parquet') test_df = spark.read.parquet('path/to/test_df.parquet')

Spark DataFrameWriter Documentation

Question 3

Question Type: **MultipleChoice**

A data scientist is developing a single-node machine learning model. They have a large number of model configurations to test as a part of their experiment. As a result, the model tuning process takes too long to complete. Which of the following approaches can be used to speed up the model tuning process?

Options:

A- Implement MLflow Experiment Tracking

B- Scale up with Spark ML

C- Enable autoscaling clusters

D- Parallelize with Hyperopt

Answer:

D

Explanation:

To speed up the model tuning process when dealing with a large number of model configurations, parallelizing the hyperparameter search using Hyperopt is an effective approach. Hyperopt provides tools like SparkTrials which can run hyperparameter optimization in parallel across a Spark cluster.

Example:

from hyperopt import fmin, tpe, hp, SparkTrials search_space = { 'x': hp.uniform('x', 0, 1), 'y': hp.uniform('y', 0, 1) } def objective(params): return params['x'] ** 2 + params['y'] ** 2 spark_trials = SparkTrials(parallelism=4) best = fmin(fn=objective, space=search_space, algo=tpe.suggest, max_evals=100, trials=spark_trials)

Hyperopt Documentation

Question 4

Question Type: **MultipleChoice**

A machine learning engineer is trying to scale a machine learning pipeline by distributing its single-node model tuning process. After broadcasting the entire training data onto each core, each core in the cluster can train one model at a time. Because the tuning process is still running slowly, the engineer wants to increase the level of parallelism from 4 cores to 8 cores to speed up the tuning process. Unfortunately, the total memory in the cluster cannot be increased.

In which of the following scenarios will increasing the level of parallelism from 4 to 8 speed up the tuning process?

Options:

A- When the tuning process in randomized

B- When the entire data can fit on each core

C- When the model is unable to be parallelized

D- When the data is particularly long in shape

E- When the data is particularly wide in shape

Answer:

B

Explanation:

Increasing the level of parallelism from 4 to 8 cores can speed up the tuning process if each core can handle the entire dataset. This ensures that each core can independently work on training a model without running into memory constraints. If the entire dataset fits into the memory of each core, adding more cores will allow more models to be trained in parallel, thus speeding up the process.

Parallel Computing Concepts

Question 5

Question Type: **MultipleChoice**

A machine learning engineer wants to parallelize the inference of group-specific models using the Pandas Function API. They have developed the apply_model function that will look up and load the correct model for each group, and they want to apply it to each group of DataFrame df.

They have written the following incomplete code block:

```
prediction_df = (df
    .groupby("device_id")
    ._____(apply_model, schema=apply_return_schema)
)
```

Which piece of code can be used to fill in the above blank to complete the task?

Options:

A- applyInPandas

B- groupedApplyInPandas

C- mapInPandas

D- predict

Answer:

A

Explanation:

To parallelize the inference of group-specific models using the Pandas Function API in PySpark, you can use the applyInPandas function. This function allows you to apply a Python function on each group of a DataFrame and return a DataFrame, leveraging the power of pandas UDFs (user-defined functions) for better performance.

prediction_df = (df.groupby('device_id') .applyInPandas(apply_model, schema=apply_return_schema))

In this code:

groupby('device_id'): Groups the DataFrame by the 'device_id' column.

applyInPandas(apply_model, schema=apply_return_schema): Applies the apply_model function to each group and specifies the schema of the return DataFrame.

PySpark Pandas UDFs Documentation

Question 6

Question Type: MultipleChoice

A data scientist is using the following code block to tune hyperparameters for a machine learning model:

```
num_evals = 4
trials = SparkTrials()
best_hyperparam = fmin(
    fn=objective_function,
    space=search_space,
    algo=tpe.suggest,
    max_evals=num_evals,
    trials=trials
)
```

Which change can they make the above code block to improve the likelihood of a more accurate model?

Options:

A- Increase num_evals to 100

B- Change fmin() to fmax()

C- Change sparkTrials() to Trials()

D- Change tpe.suggest to random.suggest

Answer:

A

Explanation:

To improve the likelihood of a more accurate model, the data scientist can increase num_evals to 100. Increasing the number of evaluations allows the hyperparameter tuning process to explore a larger search space and evaluate more combinations of hyperparameters, which increases the chance of finding a more optimal set of hyperparameters for the model.

Databricks documentation on hyperparameter tuning: Hyperparameter Tuning

Question 7

Question Type: MultipleChoice

Which statement describes a Spark ML transformer?

Options:

A- A transformer is an algorithm which can transform one DataFrame into another DataFrame

B- A transformer is a hyperparameter grid that can be used to train a model

C- A transformer chains multiple algorithms together to transform an ML workflow

D- A transformer is a learning algorithm that can use a DataFrame to train a model

Answer:

A

Explanation:

In Spark ML, a transformer is an algorithm that can transform one DataFrame into another DataFrame. It takes a DataFrame as input and produces a new DataFrame as output. This transformation can involve adding new columns, modifying existing ones, or applying feature transformations. Examples of transformers in Spark MLlib include feature transformers like StringIndexer, VectorAssembler, and StandardScaler.

Databricks documentation on transformers: Transformers in Spark ML

Question 8

Question Type: MultipleChoice

A machine learning engineer is using the following code block to scale the inference of a single-node model on a Spark DataFrame with one million records:

```
@pandas_udf("double")
def predict(iterator: Iterator[pd.DataFrame]) -> Iterator[pd.Series]:
    model_path = f"runs:/{run.info.run_id}/model"
    model = mlflow.sklearn.load_model(model_path)
    for features in iterator:
        pdf = pd.concat(features, axis=1)
        yield pd.Series(model.predict(pdf))
```

Assuming the default Spark configuration is in place, which of the following is a benefit of using an Iterator?

Options:

A- The data will be limited to a single executor preventing the model from being loaded multiple times

B- The model will be limited to a single executor preventing the data from being distributed

C- The model only needs to be loaded once per executor rather than once per batch during the inference process

D- The data will be distributed across multiple executors during the inference process

Answer:

C

Explanation:

Using an iterator in the pandas_udf ensures that the model only needs to be loaded once per executor rather than once per batch. This approach reduces the overhead associated with repeatedly loading the model during the inference process, leading to more efficient and faster predictions. The data will be distributed across multiple executors, but each executor will load the model only once, optimizing the inference process.

Databricks documentation on pandas UDFs: Pandas UDFs

Question 9

Question Type: MultipleChoice

A data scientist has developed a linear regression model using Spark ML and computed the predictions in a Spark DataFrame preds_df with the following schema:

prediction DOUBLE

actual DOUBLE

Which of the following code blocks can be used to compute the root mean-squared-error of the model according to the data in preds_df and assign it to the rmse variable?

A)

```
rmse = RegressionEvaluator(
    predictionCol="prediction",
    labelCol="actual",
    metricName="rmse"
)
```

B)

```
rmse = BinaryClassificationEvaluator(
    predictionCol="prediction",
    labelCol="actual",
    metricName="rmse"
)
```

C)

```
regression_evaluator = RegressionEvaluator(
    predictionCol="prediction",
    labelCol="actual",
    metricName="rmse"
)
rmse = regression_evaluator.evaluate(preds_df)
```

D)

```
classification_evaluator = BinaryClassificationEvaluator(
    predictionCol="prediction",
    labelCol="actual",
    metricName="rmse"
)
rmse = classification_evaluator.evaluate(preds_df)
```

Options:

A- Option A

B- Option B

C- Option C

D- Option D

Answer:

C

Explanation:

To compute the root mean-squared-error (RMSE) of a linear regression model using Spark ML, the RegressionEvaluator class is used. The RegressionEvaluator is specifically designed for regression tasks and can calculate various metrics, including RMSE, based on the columns containing predictions and actual values.

The correct code block to compute RMSE from the preds_df DataFrame is:

regression_evaluator = RegressionEvaluator(predictionCol='prediction', labelCol='actual', metricName='rmse') rmse = regression_evaluator.evaluate(preds_df)

This code creates an instance of RegressionEvaluator, specifying the prediction and label columns, as well as the metric to be computed ('rmse'). It then evaluates the predictions in preds_df and assigns the resulting RMSE value to the rmse variable.

Options A and B incorrectly use BinaryClassificationEvaluator, which is not suitable for regression tasks. Option D also incorrectly uses BinaryClassificationEvaluator.

PySpark ML Documentation