Download Databricks-Machine-Learning-Associate Exam Dumps Free

Shared by Austin on 17-06-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

An organization is developing a feature repository and is electing to one-hot encode all categorical feature variables. A data scientist suggests that the categorical feature variables should not be one-hot encoded within the feature repository.

Which of the following explanations justifies this suggestion?

Options:

A- One-hot encoding is a potentially problematic categorical variable strategy for some machine learning algorithms.
B- One-hot encoding is dependent on the target variable's values which differ for each apaplication.
C- One-hot encoding is computationally intensive and should only be performed on small samples of training sets for individual machine learning problems.
D- One-hot encoding is not a common strategy for representing categorical feature variables numerically.

Answer:

A

Explanation:

The suggestion not to one-hot encode categorical feature variables within the feature repository is justified because one-hot encoding can be problematic for some machine learning algorithms. Specifically, one-hot encoding increases the dimensionality of the data, which can be computationally expensive and may lead to issues such as multicollinearity and overfitting. Additionally, some algorithms, such as tree-based methods, can handle categorical variables directly without requiring one-hot encoding.

Databricks documentation on feature engineering: Feature Engineering

Question 2

Question Type: MultipleChoice

Which of the following tools can be used to parallelize the hyperparameter tuning process for

single-node machine learning models using a Spark cluster?

Options:

A- MLflow Experiment Tracking
B- Spark ML
C- The process will be unable to parallelize tuning due to the distributed nature of pipeline
D- Autoscaling clusters
E- Delta Lake

Answer:

B

Explanation:

Spark ML (part of Apache Spark's MLlib) is designed to handle machine learning tasks across multiple nodes in a cluster, effectively parallelizing tasks like hyperparameter tuning. It supports various machine learning algorithms that can be optimized over a Spark cluster, making it suitable for parallelizing hyperparameter tuning for single-node machine learning models when they are adapted to run on Spark.

Reference

Apache Spark MLlib Guide: https://spark.apache.org/docs/latest/ml-guide.html

Spark ML is a library within Apache Spark designed for scalable machine learning. It provides tools to handle large-scale machine learning tasks, including parallelizing the hyperparameter tuning process for single-node machine learning models using a Spark cluster. Here's a detailed explanation of how Spark ML can be used:

Hyperparameter Tuning with CrossValidator: Spark ML includes the CrossValidator and TrainValidationSplit classes, which are used for hyperparameter tuning. These classes can evaluate multiple sets of hyperparameters in parallel using a Spark cluster.

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

from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Define the model

model = ...

# Create a parameter grid

paramGrid = ParamGridBuilder() \
```

```
.addGrid(model.hyperparam1, [value1, value2]) \

.addGrid(model.hyperparam2, [value3, value4]) \

.build()

# Define the evaluator

evaluator = BinaryClassificationEvaluator()

# Define the CrossValidator

crossval = CrossValidator(estimator=model,

estimatorParamMaps=paramGrid,

evaluator=evaluator,

numFolds=3)
```

Parallel Execution: Spark distributes the tasks of training models with different hyperparameters across the cluster's nodes. Each node processes a subset of the parameter grid, which allows multiple models to be trained simultaneously.

Scalability: Spark ML leverages the distributed computing capabilities of Spark. This allows for efficient processing of large datasets and training of models across many nodes, which speeds up the hyperparameter tuning process significantly compared to single-node computations.

Reference

Apache Spark MLlib Documentation

Hyperparameter Tuning in Spark ML

Question 3

Question Type: MultipleChoice

The implementation of linear regression in Spark ML first attempts to solve the linear regression problem using matrix decomposition, but this method does not scale well to large datasets with a large number of variables.

Which of the following approaches does Spark ML use to distribute the training of a linear regression model for large data?

Options:

A- Logistic regression
B- Spark ML cannot distribute linear regression training
C- Iterative optimization
D- Least-squares method
E- Singular value decomposition

Answer:

C

Explanation:

For large datasets with many variables, Spark ML distributes the training of a linear regression model using iterative optimization methods. Specifically, Spark ML employs algorithms such as Gradient Descent or L-BFGS (Limited-memory Broyden--Fletcher--Goldfarb--Shanno) to iteratively minimize the loss function. These iterative methods are suitable for distributed computing environments and can handle large-scale data efficiently by partitioning the data across nodes in a cluster and performing parallel updates. Reference:

Spark MLlib Documentation (Linear Regression with Iterative Optimization).

Question 4

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 option best 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 5

Question Type: MultipleChoice

Which of the following machine learning algorithms typically uses bagging?

Options:

A- IGradient boosted trees
B- K-means
C- Random forest
D- Decision tree

Answer:

C

Explanation:

Random Forest is a machine learning algorithm that typically uses bagging (Bootstrap Aggregating). Bagging is a technique that involves training multiple base models (such as decision trees) on different subsets of the data and then combining their predictions to improve overall model performance. Each subset is created by randomly sampling with replacement from the original dataset. The Random Forest algorithm builds multiple decision trees and merges them to get a more accurate and stable prediction.

Databricks documentation on Random Forest: Random Forest in Spark ML

Question 6

Question Type: MultipleChoice

A machine learning engineer would like to develop a linear regression model with Spark ML to predict the price of a hotel room. They are using the Spark DataFrame train_df to train the model.

The Spark DataFrame train_df has the following schema:

```
hotel_room_id STRING,
price DOUBLE,
features UDT
```

The machine learning engineer shares the following code block:

```
lr = LinearRegression(featuresCol="features", labelCol="price")
lr_model = lr.fit(train_df)
```

Which option best changes does the machine learning engineer need to make to complete the task?

Options:

A- They need to call the transform method on train df
B- They need to convert the features column to be a vector
C- They do not need to make any changes
D- They need to utilize a Pipeline to fit the model
F- They need to split the features column out into one column for each feature

Answer:

B

Explanation:

In Spark ML, the linear regression model expects the feature column to be a vector type. However, if the features column in the DataFrame train_df is not already in this format (such as being a column of type UDT or a non-vectorized type), the engineer needs to convert it to a vector column using a transformer like VectorAssembler. This is a critical step in preparing the data for modeling as Spark ML models require input features to be combined into a single vector column.

Reference

Spark MLlib documentation for LinearRegression:
https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-regression

Question 7

Question Type: MultipleChoice

A data scientist has replaced missing values in their feature set with each respective feature variable's median value. A colleague suggests that the data scientist is throwing away valuable information by doing this.

Which of the following approaches can they take to include as much information as possible in the feature set?

Options:

A- Impute the missing values using each respective feature variable's mean value instead of the median value
B- Refrain from imputing the missing values in favor of letting the machine learning algorithm determine how to handle them
C- Remove all feature variables that originally contained missing values from the feature set
D- Create a binary feature variable for each feature that contained missing values indicating whether each row's value has been imputed
E- Create a constant feature variable for each feature that contained missing values indicating the percentage of rows from the feature that was originally missing

Answer:

D

Explanation:

By creating a binary feature variable for each feature with missing values to indicate whether a value has been imputed, the data scientist can preserve information about the original state of the data. This approach maintains the integrity of the dataset by marking which values are original and which are synthetic (imputed). Here are the steps to implement this approach:

Identify Missing Values: Determine which features contain missing values.

Impute Missing Values: Continue with median imputation or choose another method (mean, mode, regression, etc.) to fill missing values.

Create Indicator Variables: For each feature that had missing values, add a new binary feature. This feature should be '1' if the original value was missing and imputed, and '0' otherwise.

Data Integration: Integrate these new binary features into the existing dataset. This maintains a record of where data imputation occurred, allowing models to potentially weight these observations differently.

Model Adjustment: Adjust machine learning models to account for these new features, which might involve considering interactions between these binary indicators and other features.

Reference

'Feature Engineering for Machine Learning' by Alice Zheng and Amanda Casari (O'Reilly Media, 2018), especially the sections on handling missing data.

Scikit-learn documentation on imputing missing values: https://scikit-learn.org/stable/modules/impute.html

Question 8

Question Type: MultipleChoice

A machine learning engineering team has a Job with three successive tasks. Each task runs a single notebook. The team has been alerted that the Job has failed in its latest run.

Which of the following approaches can the team use to identify which task is the cause of the failure?

Options:

A- Run each notebook interactively
B- Review the matrix view in the Job's runs
C- Migrate the Job to a Delta Live Tables pipeline
D- Change each Task's setting to use a dedicated cluster

Answer:

B

Explanation:

To identify which task is causing the failure in the job, the team should review the matrix view in the Job's runs. The matrix view provides a clear and detailed overview of each task's status, allowing the team to quickly identify which task failed. This approach is more efficient than running each notebook interactively, as it provides immediate insights into the job's execution flow and any issues that occurred during the run.

Databricks documentation on Jobs: Jobs in Databricks