Data Engineering for Machine Learning: Comprehensive Review Questions and Answers

Based on DSC 208R – Data Management for Analytics by Arun Kumar, UC San Diego

June 2025

MapReduce Review Questions

1. **Briefly explain 2 reasons why parallel RDBMSs did not suffice for parallel processing of "Big Data."**

   a) **Developability:** Custom data models and computations were hard to program on SQL/RDBMSs, leading to a need for simpler APIs.

   b) **Fault Tolerance:** RDBMSs were not designed for the graceful handling of worker failures at the scale of thousands of machines, which is common in "Big Data" environments. Other reasons include elasticity and cost.

2. **What is MapReduce? What is Hadoop? How are they different?**

   a) **MapReduce:** A programming model and an associated implementation for processing and generating large datasets with a parallel, distributed algorithm on a cluster. It defines two main functions: Map and Reduce.

   b) **Hadoop:** A Free and Open-Source Software (FOSS) system that implements the MapReduce programming model and includes HDFS (Hadoop Distributed File System) as its underlying filesystem.

   c) **Difference:** MapReduce is a programming model/paradigm, while Hadoop is a specific software framework that provides an implementation of the MapReduce model along with a distributed file system. Hadoop handles aspects like input splits, data distribution, shuffling, and fault tolerance automatically for MapReduce jobs.

3. **Can the data types of the inputs to Map and Reduce differ?** Yes, the data types of the inputs to Map and Reduce functions can differ. The Map function takes an input (e.g., a document name and text) and emits key-value pairs (e.g., (word, 1)). The Reduce function then takes a key and an iterator of values emitted by the Map phase and produces an output (e.g., (word, sum of counts)).

4. **Which extended relational algebra operator does MapReduce resemble the most?** MapReduce most closely resembles the **Group By and Aggregate** operator from extended relational algebra. The Map phase often processes records and emits keys for grouping, while the Reduce phase aggregates values within each group.

5. **How does Hadoop/MapReduce achieve fault tolerance?** Hadoop/MapReduce achieves fault tolerance by automatically handling worker failures. If a worker fails during a Map or Reduce task, the system detects the failure and re-executes the failed task on another available worker. This is facilitated by replicating data blocks in HDFS and by the idempotent nature of many MapReduce tasks.

6. **What are Map-only jobs in MapReduce? Why do they arise?**

   a) **Map-only jobs:** These are MapReduce jobs where no Reduce phase is necessary. The entire computation can be completed by the Map tasks alone, often because no aggregation or grouping across different keys is required.

b) **Why they arise:** They typically arise when the task involves a row-wise or element-wise transformation or filtering operation on the data, where each input record can be processed independently without needing to combine results based on a key (e.g., projection, selection, or per-record feature extraction like Bag of Words).

7. **Suppose you are given a large relation stored as an HDFS file. You are asked to compute 3 AVG values on 3 different attributes. Write a single MapReduce job to compute the above.** To compute 3 AVG values on 3 different attributes (say, $attr1$, $attr2$, $attr3$) in a single MapReduce job:

   Map Function: For each tuple/record in the HDFS file:

   Emit a global dummy key (e.g., "GLOBAL_AVG") and a value containing a tuple of:

   $$(attr1_{value}, attr2_{value}, attr3_{value}, 1)$$

   The 1 acts as a count for that record.

```
    def map(key, record): # record is a tuple/row
        attr1_val = record.get('attr1', 0)
        attr2_val = record.get('attr2', 0)
        attr3_val = record.get('attr3', 0)
        emit("GLOBAL_AVG", (attr1_val, attr2_val, attr3_val, 1))
```

   Reduce Function: For the global dummy key, the reducer receives an iterator of all emitted $(attr1_{value}, attr2_{value}, attr3_{value}, 1)$ tuples. It sums up all $attr1_{value}$, $attr2_{value}$, $attr3_{value}$ and counts the total number of records (sum of '1's). Then, it computes the average for each attribute and emits the final result.

```
    def reduce(key, list_of_values): # key is "GLOBAL_AVG"
        total_attr1 = 0
        total_attr2 = 0
        total_attr3 = 0
        count = 0
        for val_tuple in list_of_values:
            total_attr1 += val_tuple[0]
            total_attr2 += val_tuple[1]
            total_attr3 += val_tuple[2]
            count += val_tuple[3] # Should always be 1 for this design

        avg_attr1 = total_attr1 / count if count > 0 else 0
        avg_attr2 = total_attr2 / count if count > 0 else 0
        avg_attr3 = total_attr3 / count if count > 0 else 0

        emit(key, (avg_attr1, avg_attr2, avg_attr3))
```

Feature Engineering Review Questions

1. **Briefly explain 2 reasons why the model building stage of the data science lifecycle can become challenging.**

   a) **Iterative Nature and Hyper-parameter Tuning:** Finding the optimal model configuration often requires extensive iterative experimentation, involving selecting algorithms, engineering features, and tuning numerous hyper-parameters. This trial-and-error process can be time-consuming and computationally expensive.

   b) **Feature Engineering Complexity:** Transforming raw data into effective features can be difficult, time-consuming, and requires significant domain expertise. Poor features can severely limit model performance regardless of the chosen algorithm.

2. **What are the 3 decisions inherent to the model selection process?** The three decisions inherent to the model selection process are:

   a) **Feature Engineering (FE):** Deciding which features to use, create, or transform from the raw data.

   b) **Algorithm Selection (AS):** Choosing the appropriate machine learning algorithm(s) for the task (e.g., Logistic Regression, XGBoost, Neural Networks).

   c) **Hyper-Parameter Tuning (HT):** Selecting the optimal values for the model's hyper-parameters (parameters that are set before training begins).

3. **Give an example of a feature engineering step that can be scaled using a Map-only job.** An example of a feature engineering step that can be scaled using a Map-only job is **Bag of Words (BoW)** or **N-gram extraction** from a text corpus. Each document can be processed independently by a Mapper to count word frequencies or N-gram occurrences without needing to communicate with other Mappers or a Reduce step.

4. **Why is hyper-parameter tuning needed? Name 2 tuning methods.**

   a) **Why needed:** Hyper-parameter tuning is needed because hyper-parameters act as "knobs" for an ML model or its training algorithm, controlling the bias-variance trade-off in a dataset-specific manner. Optimizing them is crucial for a model to learn effectively and achieve good performance on unseen data.

   b) **Two tuning methods:**

   - **Grid Search:** Systematically evaluates all possible combinations of hyper-parameter values from a predefined discrete set.
   - **Random Search:** Samples hyper-parameter values randomly from defined distributions or ranges, often more efficient than grid search in high-dimensional spaces.

5. **Briefly explain 1 pro and 1 con of AutoML tools.**

   a) **Pro:** AutoML tools can significantly **reduce the manual effort and time** required for tasks like feature engineering, algorithm selection, and hyper-parameter tuning, making ML more accessible to non-experts and accelerating the development process.

   b) **Con:** AutoML tools can be **computationally intensive** due to the extensive search space they explore. They might also lead to less interpretability or understanding of the underlying model choices compared to manual approaches.

6. **Suppose you decide to try logistic regression and XGBoost for a tabular data classification task. You try 4 learning rates for each model, as well as 2 regularizer values for the logistic model and 3 values of number of trees for XGBoost. What is the total number of models trained in this model selection process?** Let's calculate the number of models for each:

   - **Logistic Regression:**
     - Learning rates: 4
     - Regularizer values: 2
     - Total Logistic Regression models $= 4 \times 2 = 8$
   - **XGBoost:**
     - Learning rates: 4
     - Number of trees: 3
     - Total XGBoost models $= 4 \times 3 = 12$

   Total number of models trained = Total Logistic Regression models + Total XGBoost models $= 8 + 12 =$ **20** models.

Spark and Dataflow Systems Review Questions

1. **Briefly explain 2 differences between Hadoop and Spark.**

   a) **In-memory Processing vs. Disk-based:** Spark primarily exploits distributed memory to cache data between computations, leading to significantly faster iterative algorithms and interactive queries. Hadoop MapReduce, by contrast, heavily relies on disk I/O between stages.

   b) **Fault Tolerance Mechanism:** Spark's fault tolerance is based on "lineage," where it records the transformations applied to data (DAG of operations). If a partition is lost, Spark can recompute it from its lineage. Hadoop's fault tolerance primarily relies on data replication (HDFS) and re-executing failed tasks.

2. **What is the relationship between Spark's RDD API, MapReduce, and extended relational algebra?** Spark's RDD (Resilient Distributed Dataset) API is a low-level API that forms the core abstraction for distributed data processing. It provides transformations (like *map*, $filter$, $reduceByKey$) that generalize operations found in both MapReduce (e.g., *map* is similar to Map, $reduceByKey$ is similar to Reduce) and extended relational algebra (e.g., $filter$ for selection, *map* for projection). Essentially, RDDs subsume most of the operations possible with MapReduce and relational algebra, offering a more flexible and unified dataflow programming model.

3. **What is SparkSQL? What is its relationship to RDD API?**

   a) **SparkSQL:** A Spark module for working with structured data. It provides a programming interface for Spark that allows users to query data using SQL or a DataFrame/Dataset API.

   b) **Relationship to RDD API:** SparkSQL is built on top of the RDD API. DataFrames and Datasets in SparkSQL are essentially distributed collections of rows with a schema, which are implemented as RDDs internally. SparkSQL leverages Spark's Catalyst Optimizer to optimize queries, often compiling them down to efficient RDD operations.

4. **Why does Databricks recommend using SparkSQL/DataFrame API instead of RDD API nowadays?** Databricks (the company behind Spark) recommends using SparkSQL/DataFrame API over the lower-level RDD API for several reasons:

   - **Optimization:** DataFrames carry schema information, allowing Spark's Catalyst Optimizer to perform significant query optimizations (e.g., predicate pushdown, column pruning) that are not possible with RDDs.
   - **Ease of Use:** DataFrame/Dataset API is more user-friendly and expressive for common data manipulation tasks, resembling Pandas DataFrames.
   - **Performance:** Due to optimizations, DataFrames often yield better performance for structured data operations.

5. **Briefly describe 2 query optimizations implementing in SparkSQL.**

   a) **Predicate Pushdown:** This optimization pushes filtering conditions (WHERE clauses) as close to the data source as possible. This reduces the amount of data that needs to be read and shuffled, leading to significant performance improvements.

   b) **Column Pruning:** This optimization identifies and removes columns that are not needed for the query result. By selecting only the necessary columns from the source, it reduces the amount of data processed, transferred over the network, and stored in memory. Other optimizations include join reordering, common subexpression elimination, and code generation.

6. **What is a "data lakehouse"?** A "data lakehouse" is a new paradigm in data architecture that combines the best features of data lakes and data warehouses. It aims to provide the flexibility, scalability, and low cost of data lakes (which store raw, unstructured, and semi-structured data) with the data management features, schema enforcement, transaction support, and performance of data warehouses. This allows for both traditional BI and advanced analytics (ML) on the same platform.

7. **What is the difference between Koalas and DataFrame on Spark?**

a) **Spark DataFrame:** Spark's native high-level API for structured data, offering distributed data processing capabilities.

b) **Koalas (now part of PySpark):** A Python library that implements the pandas API on top of Apache Spark. Its primary purpose is to make it easier for pandas users to transition to Spark for big data tasks by providing a familiar API while leveraging Spark's distributed processing power.

c) **Difference:** Koalas is essentially a wrapper or an implementation of the pandas API that uses Spark DataFrames under the hood. While a Spark DataFrame is the core distributed data structure in Spark's high-level API, Koalas provides a syntax and functionality that mirrors pandas, allowing users to write pandas-like code that executes on a Spark cluster.