Sakshi Manish Mukkirwar

sakshimanish.mukkirwar@sjsu.edu

DATA 228 – Big data tech

Databricks Implementation

**Task 1: Implement a PySpark program to list out the Wikipedia pages with the top ten PageRanks on a sample Wikipedia XML dataset. This exercise will help you understand how to process XML data, extract relevant information, and perform PageRank calculations using PySpark.**

Solution:

Calculation1:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

In THIS CODE

1. Created a function that uses a regular expression to extract wiki links from the text.

and converts the Data Frame to RDD.

1. Then, applied transformations to extract links, count link occurrences, and format the data.
2. Then we convert the RDD of links back to a data frame with two columns: "page" and "out\_links."
3. Initialize PageRank: Created a new column in the data Frame to initialize PageRank values.
4. Iteratively Calculate PageRank: Performed PageRank calculations iteratively for 10 iterations, considering link contributions and applying a damping factor.
5. Ordered the pages by PageRank and displayed the top 10 pages with the highest ranks.

Calculation2:

A screenshot of a computer

Description automatically generated

In this page rank algorithm:

1. We use a small dataset where each URL (A, B, C, D) links to its neighbors.
2. We initialize each URL's rank to 1.0.
3. Over 10 iterations, we compute the contribution of each URL to its neighbors' ranks.
4. We update each URL's rank by combining its neighbors' contributions.
5. The final ranks are printed after 10 iterations.

This is a basic PageRank algorithm implemented with RDDs in PySpark. Adjustments might be needed for larger datasets and more advanced use cases.

**Task 2:**

* 1. **Explain the code to the extent you can.**

 This script is a basic implementation of logistic regression using Apache Spark and NumPy. It's designed to work on large datasets in a distributed environment. Here's a breakdown of the key parts of the code:

1. Import Statements and Constants:
   * The script imports necessary modules from Python's standard library, NumPy for numerical computations, and PySpark for distributed computing.
   * D is defined as a constant representing the number of dimensions (features) for each data point.
2. Reading Batches:
   * readPointBatch is a function that takes strings (lines from a file) that iterate and converts them into a NumPy matrix. Each line represents a data point with D features plus a label.
   * The data is expected in a specific format: <label> <x1> <x2> ... <xD>, where <label> is the target variable (0 or 1 for logistic regression) and <xi> are the feature values.
3. Main Execution Guard:
   * The if \_\_name\_\_ == "\_\_main\_\_": block ensures that the following code only runs when the script is executed directly, not when imported as a module.
4. Argument Checking:
   * The script expects two command-line arguments: the path to the input data file and the number of iterations to run the logistic regression algorithm. It checks for this and exits if the correct arguments are not provided.
5. Spark Session:
   * A Spark session is initialized. This is the entry point for using Spark functionalities.
6. Data Loading and Caching:
   * The input data file is read as a text file into an RDD (Resilient Distributed Dataset), which is a fundamental data structure of Spark.
   * The data is then partitioned into batches using the readPointBatch function and cached. Caching is used to improve performance, as the data will be accessed multiple times during the iterations.
7. Model Initialization:
   * The weights w for the logistic regression model are initialized to random values.
8. Gradient Computation:
   * The gradient function computes the gradient of the logistic loss function for a batch of points. It applies the logistic function to the weighted sum of inputs and calculates the gradient based on the difference between the predicted and actual labels.
   * The add function is used to sum gradients across different partitions of the RDD.
9. Model Training:
   * The script performs the specified number of iterations of gradient descent. In each iteration, it updates the weights w by subtracting a portion of the gradient (computed by the gradient function and aggregated using the reduce operation with the add function).
10. Final Output:
    * After all iterations, it prints the final weights.
11. Spark Session Termination:
    * Finally, the script stops the Spark session.

This script demonstrates distributed computing principles where data is partitioned across different nodes of a cluster and processed in parallel, which is useful for large datasets that cannot fit into the memory of a single machine.

* 1. **Make any changes to suit the task and run the code on one of your favorite classification datasets. You can use spark installed on your laptop. Determine the train and test accuracy.**

**Dataset Used:** Breast Cancer

A screenshot of a computer

Description automatically generated

**A screenshot of a computer program

Description automatically generated**

* 1. **Use LogisticRegression from pyspark.ml.classification to determine the train and test accuracy on the same dataset. Compare the two approaches.**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**Task 3:**

1. **Provide an algorithm to handle the case where the stream to a DGIM algorithm is not bits, but integers, and we want the sum of the last *k.***

Algorithm to handle the stream of integers:

1. The original DGIM algorithm is used for approximating the count of '1's in the last k bits of a stream. We're now looking to extend this idea to handle the sum of the last k integers in a stream.
2. Buckets: Instead of storing '1's count, our bucket will store the sum of integers.
3. Bucket Rules:
   1. Each bucket contains the sum of a contiguous segment of the stream.
   2. Buckets of the same size (i.e., covering the same number of stream elements) are adjacent.
   3. You can have at most 2 buckets of the same size.
4. Merging: When you get a third bucket of the same size:
   1. Merge the oldest two buckets.
   2. The sum in the new bucket becomes the sum of the two merged buckets.
5. Eviction: If a bucket's oldest value is outside the window of the last k numbers, remove it.
6. Approximating Sum: To approximate the sum of the last k numbers, sum up the values in the buckets that are fully within the last k numbers and add a fraction of the bucket that spans the boundary.
7. **Write a program in Python to handle the case where the stream to a DGIM algorithm is not bits, but integers, and we want the sum of the last k.**

Python Code

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

1. **Use Apache Spark and Kafka to implement it.**

Spark Application:

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**Task 4:**

* 1. Created bigquery dataset *dataset\_1* from big query public datasets and loaded them to my bucket and my dataset as shown below.

**A screenshot of a computer

Description automatically generated**

* 1. Data pulled and successfully load it on databricks.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. PySpark Calculations:
     1. Summary Statistics

A screenshot of a computer

Description automatically generated

* + 1. Number of Cases by Country

A screenshot of a computer

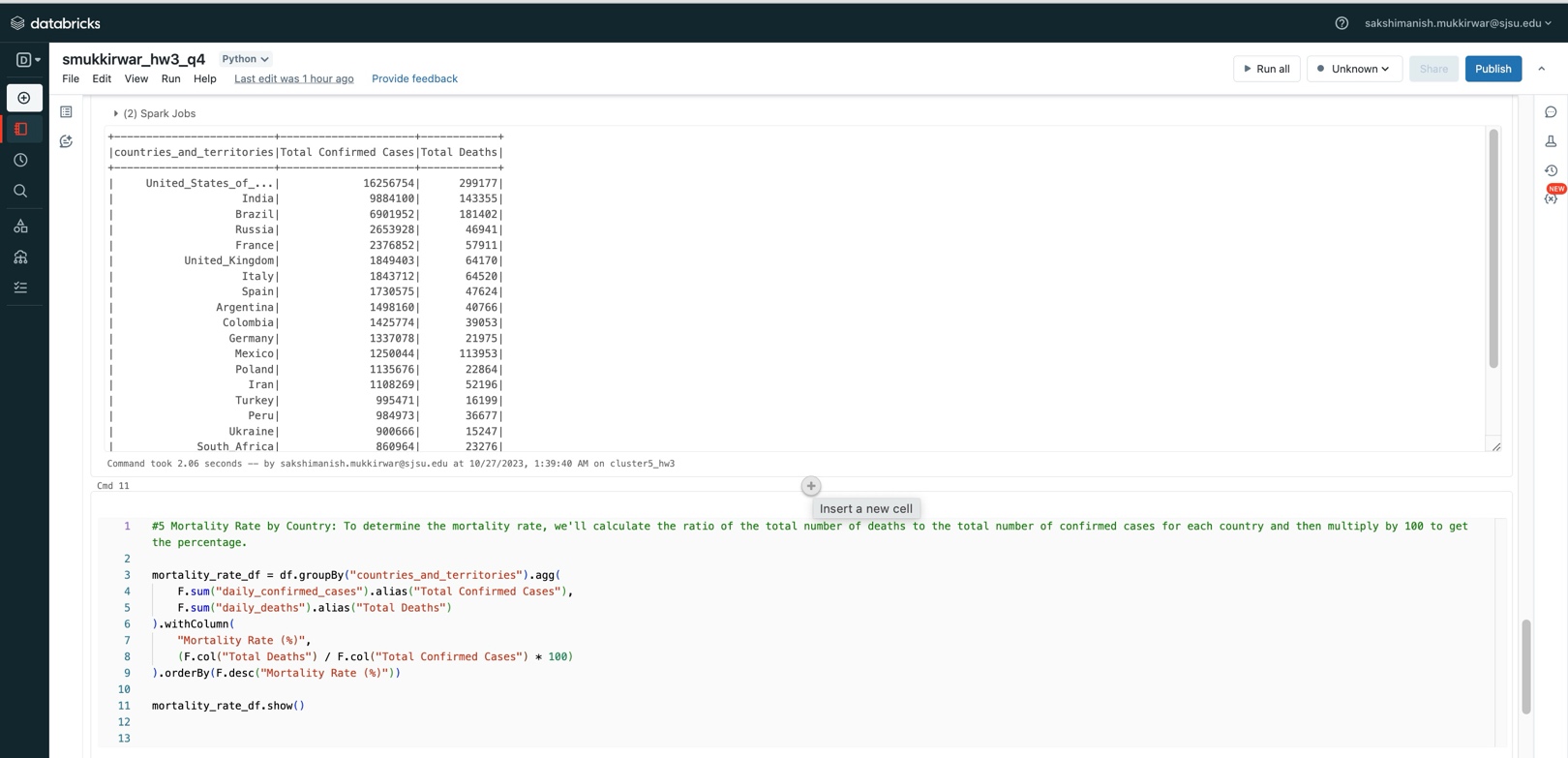
Description automatically generated

* + 1. Filtering Data

A screenshot of a chat

Description automatically generated

* + 1. Deaths vs Confirmed Cases by Country



* + 1. Mortality Rate by Country

A screenshot of a computer

Description automatically generated

* 1. Writing the output again to BigQuery.

A screenshot of a computer

Description automatically generated