Project 1 - Introduction to Apache Spark
TDT4305 - Big Data Architecture (Spring 2024)

DUE: **February 16, 2024 11:59 PM**

Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism.
  - You are allowed to talk with other students on homework assignments.
  - You can share ideas but not code, you must submit your code individually or as a group of two.
  - All submitted codes will be compared against all codes submitted this semester and in previous semesters.
- **Programming:**
  - You are required to use Python for the first four (4) tasks in this assignment.
  - Task 5 should be implemented using Scala on your local Ubuntu machine.
  - Implement the missing code under ## *To-do!* in the notebooks.
  - Ensure the notebook runs without any errors, then save the notebooks for submission.
  - Use the markdown function for writing in the notebooks.
- **Submissions:**
  - Note that if you submit your work multiple times, the most recent submission will be graded.
  - You can do this project individually or in a team of two people; if you are in a team, both members of the team must submit the same work separately on Blackboard.
  - Implement the missing code under ## *To-do!* in the notebooks.
  - Make sure that the notebook runs without any errors.
  - Only include the notebook files in the .zip files; do not include anything else such as dataset or binary files with the notebooks.
  - When you are done, download and place the notebooks and screenshot in a single .zip file named 'Project1_Spark.zip' and upload the .zip file on BlackBoard via the submission link for the assignment. Also, include the names of team members in each notebook (or only your name if you are working on the assignment individually).

  Important: *Post all questions on Piazza and only send me an email in case of (rare) anomalies!*

Databricks Setup

The first 4 tasks of this assignment will be completed on the Databrick Community Edition Environment. To run our notebooks on the Databricks platform, we need to upload the dataset files in the databricks_data folder. To set up Databricks with the data and notebooks, do the following.

1. Login into the Databricks Community Edition environment
2. Import the dataset files located in the databricks_data folder into the environment.
   N.B: The files have been compressed to csv.gz extension upload to Databricks in this format
3. Similarly import each notebook into Databricks Community Edition notebook environment

Spark Setup (Linux)

Task 5 of this assignment is intended to provide a high-level overview of Spark MLib. We need Spark to work on our local Linux environment to complete this task. The train fermentation data should be saved from the linux_data folder. Follow the steps below to install Spark on Linux and save the data.

1. Download Spark with "wget https://downloads.apache.org/spark/spark-3.3.4/spark-3.3.4-bin-hadoop3.tgz" and then unpack using" tar xfz filename" on the file.

2. Write the following commands to move the Spark software files to the respective directory (/usr/local/spark).

   # cd Downloads/
   # sudo mv spark-3.3.4-bin-hadoop3 /usr/local/spark

3. At this stage, you should be able to run Spark successfully from the spark directory using *./spark-3.3.4-bin-hadoop3/bin/spark-shell*
   However, this method has the disadvantage of only working from the spark directory. To run Spark from anywhere in the terminal, we need to setup our environment to acknowledge Spark.

4. To the ~/.bashrc file, add the following command.

   export PATH=$PATH:/usr/local/spark/bin

   exit then *$ source /.bashrc*

5. To verify Spark installation, open the Spark shell using the following command

   $ spark-shell

6. Download and mv linux_data folder to your desired location in preparation for Task 5

1 Task 1 (*20pts*)

1.1 Subtask 1: Defining the schema for the data (*7.5pts*)

A schema defines the structure and data types, ensuring that the data conforms to a specific format. This assignment's first subtask is defining schema before loading the data in the Spark cluster. By defining the schema for *badges*, we have provided a sample of what we expect for the other data as shown in the notebook. Defining schema helps prevent the insertion of incorrect or incompatible data types or for the management of evolving datasets.

[*Hint: Carefully read through the 'Dataset_description.pdf' file*]

1.2 Subtask 2: Implementing two helper functions (*7.5pts*)

In this step, we want to create two auxiliary functions: load_csv, which takes a file path and schema as input and loads the specified .csv file into a Spark DataFrame, returning the DataFrame, and 'save_df,' which takes a Spark DataFrame and stores it as a Parquet file on DBFS. Note that the .csv.gz files use a TAB ($'\backslash t'$) character as the column separator and include column names in the first row.

1.3 Subtask 3: Validating the above implementations

If you have implemented the first two subtasks correctly, this test should run without failure. You don't need to do anything here. In the end, there should be four Parquet files named 'badges', 'comments', 'posts', and 'users' in '/user/hive/warehouse'.

Note that we assumed that the data for the project has already been stored on DBFS on the '/FileStore/tables/' path (as 'badges_csv.gz', 'comments_csv.gz', 'posts_csv.gz', and 'users_csv.gz'.)

[*Hint: Check the files using %fs ls /user/hive/warehouse*]

1.4 Subtask 4: Questions about Spark related concepts (*5pts*)

This subtask aims to test your understanding of Spark concepts. Refer to the notebook for the questions and enter your answers using the markdown feature on the notebook.

---

2 Task 2 (*20pts*)

2.1 Subtask 1: Implementing two helper functions for SQL queries (*5pts*)

A very crucial subcomponent of the Apache Spark is the SQL. Apache Spark has a rich set of SQL queries which makes it robust for structured and semi-structured data processing. In this task, we will implement two functions.

1. **run_query:** Takes a single Dataframe as df and applies a Spark SQL query. Returns the content of the first column and first row of the DataFrame
2. **run_query:** This is similar to the initial implementation aside from the fact that it gets two Dataframes as df1

and df2 and applies a Spark SQL query. Returns the content of the first column and first row of the DataFrame

Note that running a Spark SQL query on a DataFrame produces a DataFrame.

2.2 Subtask 2: Subtask 2: Writing Spark SQL queries (*10pts*)

Let's write our first Spark SQL query. We will perform some data manipulation using SQL queries to extract specific entries from our data. These queries serve as input to the function defined in Subtask 2 above. For details about the queries, refer to the notebook. As usual, we have implemented the query q1 to provide an idea of what we expect.

[*Hint: The first four queries are applied to a single DataFrame while the last is on two DataFrames*]

2.3 Subtask 3: Validating the above implementations

As in the case of Task 1, you don't need to do anything here. This is a test to see if your implementations are correct.

2.4 Subtask 4: Questions about Spark related concepts (*5pts*)

Refer to the notebook for the questions and enter your answers using the markdown feature on notebook.

---

3 Task 3 (*20pts*)

3.1 Subtask 1: Implementing two helper functions (*7.5pts*)

The first helper function we need to implement in this subtask in the Pearson correlation coefficient. For this, we will use the formula in equation 1

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \cdot \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{1}$$

The `compute_pearsons_r` function receives a DataFrame and two column names and returns the Pearson correlation coefficient between values of two columns.

Just a heads up: make sure to code up the `compute_pearsons_r` yourself, no shortcuts with 'DataFrame.stat.corr.' Feel free to use it later to double-check if your implementation is correct!

Next, we define the second helper function `make_tag_graph` that takes as input a DataFrame containing the records related to 'questions' and returns a DataFrame with two columns, 'u' and 'v.' For each row i in the resulting DataFrame, the record is a tuple (u_i, v_i). u_i and v_i represent distinct tags that have appeared together for a question.

3.2 Subtask 2: Writing functions to obtain nodes and edges (*7.5pts*)

In this subtask, our motivation lies in leveraging the power of data analysis and graph algorithms to gain valuable insights from our question dataset. By understanding the relationships between tags associated with these questions, we aim to uncover patterns and dependencies among distinct tags. The functions `get_nodes` and `get_edges` are designed to help us extract meaningful information about the tags.

`get_nodes` function takes the result from running `make_tag_graph` and creates a DataFrame with a column named 'id,' which includes the tags found in the tag graph.

`get_edges` function, using the result from `make_tag_graph`, generates a DataFrame with two columns: 'src' (source node) and 'dst' (destination node). These represent pairs of distinct tags that have appeared together.

Keep in mind that here, 'tag graph' refers to the DataFrame obtained from running `make_tag_graph`. Also, 'src' and 'dst' represent different tags; they are not the same.

3.3 Subtask 3: Validating the above implementations

Validate your implementation by running the tests provided in the notebook and ensure every test runs successfully.

3.4 Subtask 4: Questions about Spark related concepts (*5pts*)

Refer to the notebook to answer the Spark related questions.

---

4 Task 4 (*20pts*)

If you have made it this far, by now, you must have acquired enough skills to query multiple DataFrames and make relevant inferences on data. This task in designed to do just that.

The core function of a question and answer platform such as Stack Exchange is connecting two main user types. Namely, users who have questions in a specific area such as computer science or data science, and knowledgeable people who can answer those questions reliably. Let's call the first category of people *'knowledge seekers'* and the second one *'expert users'* or *'experts'* for short.

Here, we want to use our data to understand experts' interests. To be specific, we want to know if experts are interested in particular topics or if their interest encompasses a diverse range of topics.

To answer the above question, we will compute the correlation between a user's expertise level and how diverse the questions they have answered. The first step is to define two variables (or measures); first for *'user expertise level'* and then for *'user interest diversity'*. Then, we will use the Pearson correlation coefficient to measure the linear correlation between the two variables.

We define the variables as:

Let **Variable A** measure user expertise level. We will use the 'Reputation' column from the 'users' table to indicate a user's expertise level on the platform. According to Stack Exchange's documentation, this variable is a rough measurement of how much the community trusts a user. In other words, it is earned by convincing your peers that you know what you're talking about.

Variable B is the measure of user interest diversity. We measure the diversity of a user's interests by computing the total number of distinct tags associated with all the questions each user has answered divided by the total number of unique tags, which is 638.

Lastly, compute the Pearson correlation coefficient between **Variable A** and **Variable B**, and based on the result you've got, answer the following question:

Do expert users have specific interests, or do they have general interests?

Kindly explain your thought process and rationale behind arriving at your answer.

You should use Apache Spark API for your implementation. You can use the Spark implementation of the Pearson correlation coefficient.

[*Hint: Explore relationships between different columns in multiple datasets and join along that column. For instance, how are OwnerUserID, Id, and ParentId related in posts and users DataFrame*]

---

5 Task 5 (*20pts*)

The final task in this project is on MLib. This work is not intended to provide a detailed overview of machine learning (ML) but to introduce the Spark MLib module. Upon completing this task, the student will have acquired some skills in loading and performing simple ML regression tasks using Spark installed locally. The fun part is that the instructor, on the other hand, will have a way of knowing for sure that the students now have Apache Spark running locally on their machines.

For this task, we have been provided with a fermentation dataset with 9 independent features and a dependent variable or label named Biomass. We plan to predict the Biomass of the fermentation process using a subset of the features *("Glucose concentration", "Acetate concentration", "Ethanol concentration", "Specific oxygen uptake rate", "Specific carbon dioxide evolution rate").*

To complete this task, we need to accomplish nine (12) main steps:

1. Run *$ spark-shell* from terminal
2. Import linear regression model, vector assembler and regression evaluator from Spark MLib
3. Load `fermentation.csv` from the linux_data folder
4. Select the subset of features indicated above and the Biomass as label
5. Create a vector assembler using these 5 subsets of features as input and use the created assembler to transform the data created in step 4
6. Select the Biomass or label and features columns and perform a 70-30 train-test split on this new data
7. Create and Fit Linear regression model on train data
8. Use the fitted model to make predictions on the test data
9. Select the prediction, Biomass or label, and features columns and show the first 10 predictions as in Figure 1
10. Print the Root Mean Squared Error (RMSE)
11. Enter team names in the next line and take a screenshot of your result as in Figure 1)
12. Rename screenshot with the surnames of team members e.g. name1_name2.png and that's it!

```
scala> predictions.select("prediction", "Biomass", "features").show(10)
+-------------------+------------+--------------------+
|         prediction|     Biomass|            features|
+-------------------+------------+--------------------+
|-0.7608474422834881|-0.009564736|[33.15768,-0.0015...|
|-1.0034230127426138|  0.06767128|[33.87953,0.00101...|
|-0.7950881569626418|-0.004445348|[33.28683,-0.0019...|
| -0.759894803112223|-0.003478578|[32.89226,1.61190...|
| -0.814459232246298|     0.12512|[33.2731,-4.08312...|
|-1.0573603885176226| -0.04069711|[34.20349,0.00140...|
|-0.7672889544796266|  -0.1116635|[32.98672,2.01372...|
|-0.8725220884787639|  0.02444548|[33.46602,0.00247...|
|-1.0009018589280405|-9.119971E-4|[33.90245,-8.6740...|
|-0.8891220795256274| -0.09661581|[33.71611,8.69806...|
+-------------------+------------+--------------------+
only showing top 10 rows


scala> println(s"Root Mean Squared Error (RMSE) on test data = $rmse")
Root Mean Squared Error (RMSE) on test data = 0.4383815622900439

scala> Kazeem Shamba
```

Figure 1: *Expected linear regression results from Task 5*

We have provided code snippets of all the packages you need and options to read the .csv file below.

```scala
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.feature.VectorAssembler

// Read CSV file into a Spark DataFrame
val ferm = spark.read
  .option("header", "true")
  .option("inferSchema", "true")
  .format("csv")
  .load("file path")
```

Listing 1: *Imports and Spark Scala Code for Data Loading*

Zip this screenshot and all four (4) notebooks as Project1_Spark.zip and submit via Blackboard.

Note: We don't expect you to get an exact RMSE of 0.44. However, if your implementation is correct, you should have a value within ± 0.05 of this value.

Good luck!