Lab 8: Spark

Due date: March 18 (Monday), 11:59pm.

Lab Assignment

Assignment Preparation

This is a pair programming lab. You can pair with anyone in the class. I will admit one team of three people.

Overview

This lab consists of two parts. In Part 1 you will implement a number of Spark computations to compute answers to database queries. Part 2 will have you use Spark to implement a machine learning algorithm: the K-Nearest Neighbor classifier on the wine quality dataset.

The algorithms themselves, and their transformation to Spark will be discussed in class.

Part 1: Database Queries

You will be working with a collection of four CSV files representing the `BAKERY` database from CSC 365. This dataset records information about one month of sales from a small bakery to a list of its dedicated customers. The dataset captures the notions of a transaction (a single purchase) and market baskets (each purchase may contain more than one item). The dataset contains four files:

1. `customers.csv` - list of 20 customers of the bakery.
2. `goods.csv` - list of 50 items on the bakery's menu.

3. `receipts.csv` - list of 1000 purchases made at the bakery during one month period.

4. `items.csv` - list of actual goods purchases (for each receipt) from the bakery.

All files are uploaded to `/data/BAKERY/` directory on HDFS.

I am releasing the `README` file for this dataset. The `README` file contains detailed descriptions of each column in each file.

With this data, write Spark code that computes answers to the following questions.

1. **(Simple)** Find all chocolate-flavored items on the menu whose price is under $5.00. For each item output the flavor, the name (food type) of the item, and the price. Sort your output in descending order by price.

2. **(Simple)** Find all customers who made a purchase on October 3, 2007. Report the name of the customer (first, last). Sort the output in alphabetical order by the customer's last name. Each customer name must appear at most once.

3. **(Moderate)** Find all dates in the first half of October of 2007 (October 1 to October 15 inclusive) on which one customer made multiple purchases. Report each date exactly once, output dates sorted in ascending order.

4. **(Moderate)** Find all customers who did not make a purchase between October 14 and October 19, 2007 (inclusively). Report their first and last names sorted alphabetically by last name.

5. **(Simple)** Find all days on which *either* `ALMETA DOMKOWSKI` made a purchase, *or* someone purchased a `Gongolais Cookie`. Sort dates in chronological order. Each date shall appear exactly once.

6. **(Simple)** Report the total amount of money spent by bakery customers in October 2007 on `Cookie`s.

7. **(Moderate)** Report all days on which more than ten tarts were purchased, sorted in chronological order.

8. **(Moderate)** Find the customer(s) who spent the most on pastries in October of 2007. Report first and last name.

9. **(Moderate)** Find the customers who never purchased an eclair (`'Eclair'`) (in October of 2007). Report their first and last names in alphabetical order by last name.

10. **(Moderate)** Find the most popular (by number of pastries sold) item. Report the item (food, flavor).

11. **(Difficult)** For every customer who DID NOT make a purchase on the day of the highest revenue, report the total number of purchases (overall) the customer made and the last date of a purchase. Order the output by the total amount of purchases.

12. **(Difficult)** Output the names of all customers who made multiple purchases (more than one receipt) on the latest day in October on which they made a purchase. Report names (first, last) of the customers and the earliest day in October on which they made a purchase, sorted in chronological order.

For each query above, create a separate python file `queryX.py` where `"X"` is the query number (1 through 12). If you want to place your helper functions into a separate package, feel free to create a file `lab7Helper.py` with all necessary functions (but do not feel compelled to do it).

Each `queryX.py` must end with printing of the output.

Note: Problems labeled **Simple** cost 10 pints. Problems labeled **Moderate** cost 15 points. Problems labeled **Difficult** cost 20 points. **Your goal for this part of the assignment is to collect 100 points**. All points beyond 100 will be extra credit (although with some diminishing returns).

Part 2: K-Nearest Neighbors

Your task is to implement a Spark version of a popular K-Nearest Neighbors classifier for the wine quality data from `/data/winequality-red-fixed.csv` file on HDFS. We use the quality column in the dataset as the category label (there are six classes total for quality values 3, 4, 5, 6, 7, and 8).

We will dicuss the K-Nearest Neighbors algorithm and how to render it in Spark in class. The specifications below concern your Lab deliverable.

Computing Pairwise Distances. Create a PySpark program `distances.py` that produces, as output, a collection of pairwise distances between two rows of data in the wine quality input dataset. The output shall be a collection of rows that look as follows:

```
rowNumber1, rowNumber2, distance
```

where `rowNumber1` and `rowNumber2` are unique ids of two rows in the wine quality dataset, and `distance` is the Eucledian distance between the vectors in these two rows.

Note. The `winequality-red-fixed.csv` file does not have unique row ids in it. It is your job to adjust your RDD with an extra column (key) that stores the row Id.

Given two vectors, $row_1 = (x_1, \ldots, x_n)$ and $row_2 = (y_1, \ldots, y_n)$ the Euclidian distance between them is computed as

$$distance(row_1, row_2) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}.$$

The output of `distances.py` shall be a data file stored on HDFS.

KNN implementation. Create a PySpark program `knn.py` that takes as input the value of k for the K Nearest Neighbors algorithm, and (optionally) the location of the distance matrix for the wine dataset (you are allowed to hardcode that location in your code, just specify this in your `README` file).

Your program shall perform an *all-but-one* K-Nearest Neighbor classification of every data point in the wine dataset. (Please note, your program will have to create an RDD out of the original `winequality-red-fixed.csv` file, as part of the your solution).

The output of your program shall be as follows:

- a file `knn-wines.out` saves somewhere in your HDFS directory where each row has the following format

  `rowId, predicted-quality, true-quality`

  Here, `rowId` is the unique id of a row from the `winequality-red-fixed.csv` dataset, `predicted-quality` is the quality value predicted by the KNN algorithm, and `true-quality` is the actual quality (the value of the `quality column` from the original file).

- A printed confusion matrix that shows for each pair of predicted and true quality values, how many rows were classified that way. The confusion matrix does not need to be saved - it shall be a printed output of your program in a readable form.

- The overall accuracy of the KNN method (i.e., the percentage of data points correctly classified).

Submission

Submit all your Python programs, and a `README` file

Use `handin` to submit as follows:

```
$ handin dekhtyar lab08 <FILES>
```

Good Luck!