Apache Hadoop –

A course for undergraduates

Homework Labs, Lecture 12

Lab: Gaining Insight with Sentiment Analysis

In this optional lab, you will use Hive's text processing features to analyze customers' comments and product ratings. You will uncover problems and propose potential solutions.

IMPORTANT: Since this lab builds on the previous one, it is important that you successfully complete the previous lab before starting this lab.

Background Information

Customer ratings and feedback are great sources of information for both customers and retailers like Dualcore. However, customer comments are typically free-form text and must be handled differently. Fortunately, Hive provides extensive support for text processing.

Step #1: Analyze Numeric Product Ratings

Before delving into text processing, you will begin by analyzing the numeric ratings customers have assigned to various products.

1. Change to the directory for this lab:

```
$ cd $ADIR/exercises/sentiment
```

2. Start Hive and use the `DESCRIBE` command to remind yourself of the table's structure.

3. We want to find the product that customers like most, but must guard against being misled by products that have few ratings assigned. Run the following query to find the product with the highest average among all those with at least 50 ratings:

```
SELECT prod_id, FORMAT_NUMBER(avg_rating, 2) AS
avg_rating
    FROM (SELECT prod_id, AVG(rating) AS avg_rating,
            COUNT(*) AS num
            FROM ratings
            GROUP BY prod_id) rated
    WHERE num >= 50
    ORDER BY avg_rating DESC
    LIMIT 1;
```

4. Rewrite, and then execute, the query above to find the product with the *lowest* average among products with at least 50 ratings. You should see that the result is product ID 1274673 with an average rating of 1.10.

Step #2: Analyze Rating Comments

We observed earlier that customers are very dissatisfied with one of the products that Dualcore sells. Although numeric ratings can help identify *which* product that is, they don't tell Dualcore *why* customers don't like the product. We could simply read through all the comments associated with that product to learn this information, but that approach doesn't scale. Next, you will use Hive's text processing support to analyze the comments.

1. The following query normalizes all comments on that product to lowercase, breaks them into individual words using the `SENTENCES` function, and passes those to the `NGRAMS` function to find the five most common bigrams (two-word combinations). Run the query in Hive:

```
SELECT EXPLODE(NGRAMS(SENTENCES(LOWER(message)), 2, 5))
    AS bigrams
    FROM ratings
    WHERE prod_id = 1274673;
```

2. Most of these words are too common to provide much insight, though the word "expensive" does stand out in the list. Modify the previous query to find the five most common *trigrams* (three-word combinations), and then run that query in Hive.

3. Among the patterns you see in the result is the phrase "ten times more." This might be related to the complaints that the product is too expensive. Now that you've identified a specific phrase, look at a few comments that contain it by running this query:

```
SELECT message
     FROM ratings
     WHERE prod_id = 1274673
       AND message LIKE '%ten times more%'
     LIMIT 3;
```

You should see three comments that say, "Why does the red one cost ten times more than the others?"

4. We can infer that customers are complaining about the price of this item, but the comment alone doesn't provide enough detail. One of the words ("red") in that comment was also found in the list of trigrams from the earlier query. Write and execute a query that will find all distinct comments containing the word "red" that are associated with product ID 1274673.

5. The previous step should have displayed two comments:

   - "What is so special about red?"
   - "Why does the red one cost ten times more than the others?"

   The second comment implies that this product is overpriced relative to similar products. Write and run a query that will display the record for product ID 1274673 in the `products` table.

6. Your query should have shown that the product was a "16GB USB Flash Drive (Red)" from the "Orion" brand. Next, run this query to identify similar products:

```
SELECT *
    FROM products
    WHERE name LIKE '%16 GB USB Flash Drive%'
      AND brand='Orion';
```

   The query results show that there are three almost identical products, but the product with the negative reviews (the red one) costs about ten times as much as the others, just as some of the comments said.

   Based on the cost and price columns, it appears that doing text processing on the product ratings has helped Dualcore uncover a pricing error.

This is the end of the lab.

Lab: Data Transformation with Hive

In this lab you will create and populate a table with log data from Dualcore's Web server. Queries on that data will reveal that many customers abandon their shopping carts before completing the checkout process. You will create several additional tables, using data from a `TRANSFORM` script and a supplied UDF, which you will use later to analyze how Dualcore could turn this problem into an opportunity.

IMPORTANT: Since this lab builds on the previous one, it is important that you successfully complete the previous lab before starting this lab.

Step #1: Create and Populate the Web Logs Table

Typical log file formats are not delimited, so you will need to use the RegexSerDe and specify a pattern Hive can use to parse lines into individual fields you can then query.

1. Change to the directory for this lab:

```
$ cd $ADIR/exercises/transform
```

2. Examine the `create_web_logs.hql` script to get an idea of how it uses a RegexSerDe to parse lines in the log file (an example log line is shown in the comment at the top of the file). When you have examined the script, run it to create the table in Hive:

```
$ hive -f create_web_logs.hql
```

3. Populate the table by adding the log file to the table's directory in HDFS:

```
$ hadoop fs -put $ADIR/data/access.log
/dualcore/web_logs
```

4. Start the Hive shell in another terminal window

5. Verify that the data is loaded correctly by running this query to show the top three items users searched for on Dualcore's Web site:

```
SELECT term, COUNT(term) AS num FROM
   (SELECT LOWER(REGEXP_EXTRACT(request,
       '/search\\?phrase=(\\S+)', 1)) AS term
       FROM web_logs
       WHERE request REGEXP '/search\\?phrase=') terms
  GROUP BY term
  ORDER BY num DESC
  LIMIT 3;
```

You should see that it returns tablet (303), ram (153) and wifi (148).

Note: The `REGEXP` operator, which is available in some SQL dialects, is similar to `LIKE`, but uses regular expressions for more powerful pattern matching. The `REGEXP` operator is synonymous with the `RLIKE` operator.

Step #2: Analyze Customer Checkouts

You've just queried the logs to see what users search for on Dualcore's Web site, but now you'll run some queries to learn whether they buy. As on many Web sites, customers add products to their shopping carts and then follow a "checkout" process to complete their purchase. Since each part of this four-step process can be identified by its URL in the logs, we can use a regular expression to easily identify them:

Step	Request URL	Description

1	/cart/checkout/step1-viewcart	View list of items added to cart
2	/cart/checkout/step2-shippingcost	Notify customer of shipping cost
3	/cart/checkout/step3-payment	Gather payment information
4	/cart/checkout/step4-receipt	Show receipt for completed order

1. Run the following query in Hive to show the number of requests for each step of the checkout process:

```
SELECT COUNT(*), request
     FROM web_logs
     WHERE request REGEXP '/cart/checkout/step\\d.+'
     GROUP BY request;
```

The results of this query highlight a major problem. About one out of every three customers abandons their cart after the second step. This might mean millions of dollars in lost revenue, so let's see if we can determine the cause.

2. The log file's `cookie` field stores a value that uniquely identifies each user session. Since not all sessions involve checkouts at all, create a new table containing the session ID and number of checkout steps completed for just those sessions that do:

```
CREATE TABLE checkout_sessions AS
 SELECT cookie, ip_address, COUNT(request) AS
steps_completed
     FROM web_logs
     WHERE request REGEXP '/cart/checkout/step\\d.+'
     GROUP BY cookie, ip_address;
```

3. Run this query to show the number of people who abandoned their cart after each step:

```
SELECT steps_completed, COUNT(cookie) AS num
     FROM checkout_sessions
     GROUP BY steps_completed;
```

You should see that most customers who abandoned their order did so after the second step, which is when they first learn how much it will cost to ship their order.

Step #3: Use `TRANSFORM` for IP Geolocation

Based on what you've just seen, it seems likely that customers abandon their carts due to high shipping costs. The shipping cost is based on the customer's location and the weight of the items they've ordered. Although this information is not in the database (since the order wasn't completed), we can gather enough data from the logs to estimate them.

We don't have the customer's address, but we can use a process known as "IP geolocation" to map the computer's IP address in the log file to an approximate physical location. Since this isn't a built-in capability of Hive, you'll use a provided Python script to `TRANSFORM` the `ip_address` field from the `checkout_sessions` table to a ZIP code, as part of HiveQL statement that creates a new table called `cart_zipcodes`.

Regarding `TRANSFORM` and UDF Examples in this Exercise

During this lab, you will use a Python script for IP geolocation and a UDF to calculate shipping costs. Both are implemented merely as a simulation – compatible with the fictitious data we use in class and intended to work even when Internet access is unavailable. The focus of these labs is on how to *use* external scripts and UDFs, rather than how the code for the examples works internally.

1. Examine the `create_cart_zipcodes.hql` script and observe the following:

    a. It creates a new table called `cart_zipcodes` based on select statement.

    b. That select statement transforms the `ip_address`, `cookie`, and `steps_completed` fields from the `checkout_sessions` table using a Python script.

    c. The new table contains the ZIP code instead of an IP address, plus the other two fields from the original table.

2. Examine the `ipgeolocator.py` script and observe the following:

    a. Records are read from Hive on standard input.

    b. The script splits them into individual fields using a tab delimiter.

    c. The `ip_addr` field is converted to `zipcode`, but the `cookie` and `steps_completed` fields are passed through unmodified.

    d. The three fields in each output record are delimited with tabs are printed to standard output.

3. Run the script to create the `cart_zipcodes` table:

```
$ hive -f create_cart_zipcodes.hql
```

Step #4: Extract List of Products Added to Each Cart

As described earlier, estimating the shipping cost also requires a list of items in the customer's cart. You can identify products added to the cart since the request URL looks like this (only the product ID changes from one record to the next):

```
/cart/additem?productid=1234567
```

1. Write a HiveQL statement to create a table called `cart_items` with two fields: `cookie` and `prod_id` based on data selected the `web_logs` table. Keep the following in mind when writing your statement:

    a. The `prod_id` field should contain only the seven-digit product ID (Hint: Use the `REGEXP_EXTRACT` function)

    b. Add a `WHERE` clause with `REGEXP` using the same regular expression as above so that you only include records where customers are adding items to the cart.

2. Execute the HiveQL statement from you just wrote.

3. Verify the contents of the new table by running this query:

```
SELECT COUNT(DISTINCT cookie) FROM cart_items WHERE
prod_id=1273905;
```

Step #5: Create Tables to Join Web Logs with Product Data

You now have tables representing the ZIP codes and products associated with checkout sessions, but you'll need to join these with the products table to get the weight of these items before you can estimate shipping costs. In order to do some more analysis later, we'll also include total selling price and total wholesale cost in addition to the total shipping weight for all items in the cart.

1. Run the following HiveQL to create a table called `cart_orders` with the information:

```
CREATE TABLE cart_orders AS
   SELECT z.cookie, steps_completed, zipcode,
          SUM(shipping_wt) as total_weight,
          SUM(price) AS total_price,
          SUM(cost) AS total_cost
     FROM cart_zipcodes z
     JOIN cart_items i
       ON (z.cookie = i.cookie)
     JOIN products p
       ON (i.prod_id = p.prod_id)
    GROUP BY z.cookie, zipcode, steps_completed;
```

Step #6: Create a Table Using a UDF to Estimate Shipping Cost

We finally have all the information we need to estimate the shipping cost for each abandoned order. You will use a Hive UDF to calculate the shipping cost given a ZIP code and the total weight of all items in the order.

1. Before you can use a UDF, you must add it to Hive's classpath. Run the following command in Hive to do that:

```
ADD JAR geolocation_udf.jar;
```

2. Next, you must register the function with Hive and provide the name of the UDF class as well as the alias you want to use for the function. Run the Hive command below to associate our UDF with the alias `CALC_SHIPPING_COST`:

```
CREATE TEMPORARY FUNCTION CALC_SHIPPING_COST AS
'com.cloudera.hive.udf.UDFCalcShippingCost';
```

3. Now create a new table called `cart_shipping` that will contain the session ID, number of steps completed, total retail price, total wholesale cost, and the estimated shipping cost for each order based on data from the `cart_orders` table:

```
CREATE TABLE cart_shipping AS
   SELECT cookie, steps_completed, total_price,
total_cost,
   CALC_SHIPPING_COST(zipcode, total_weight) AS
shipping_cost
   FROM cart_orders;
```

4. Finally, verify your table by running the following query to check a record:

```
SELECT * FROM cart_shipping WHERE
cookie='100002920697';
```

This should show that session as having two completed steps, a total retail price of $263.77, a total wholesale cost of $236.98, and a shipping cost of $9.09.

Note: The `total_price`, `total_cost`, and `shipping_cost` columns in the `cart_shipping` table contain the number of cents as integers. Be sure to divide results containing monetary amounts by 100 to get dollars and cents.

This is the end of the lab.