DSC 102: Systems for Scalable Analytics

Programming Assignment 2

1 Introduction

In this assignment we will conduct data engineering for the Amazon dataset. The extracted features will be used for your next assignment, where you train a model (or models) to predict user ratings for a product.

We will be using Apache Spark and conduct the tasks on AWS EMR, Amazon's managed service for Hadoop ecosystem applications.

You will spawn an EMR cluster (several EC2 instances with EMR added) on AWS. You will then connect to the master node of the cluster and finish all the developments and tests there. You are not expected to code anything locally.

2 Dev-kit

A dev-kit consisting skeletons and other necessary files has been provided to you along with this document. When you spawn the cluster following the instructions in Section 6, this dev-kit will be prepared on your cluster's master node automatically, you do not need to manually download it.

Within the dev-kit there are several files:

```
assignment2.ipynb -- the task descriptions and a playground for your development
assignment2.py -- the deliverable of this assignment, your final file to submit
------------------------------
log4j-spark.properties
pa2_main.py
utilities.py -- these three files are necessary for your code to run. Do not modify any of them
```

3 Dataset Description

You are expected to extract features from three tables, their schemas and descriptions are listed below:

```
1. product
    |-- asin: string, the product id, e.g., 'B00I8HVV6E'
    |-- salesRank: map, a map between category and sales rank, e.g., {'Home & Kitchen': 796318}
    |    |-- key: string, category, e.g., 'Home & Kitchen'
    |    |-- value: integer, rank, e.g., 796318
    |-- categories: array, list of list of categories, e.g., [['Home & Kitchen', 'Artwork']]
    |    |-- element: array, list of categories, e.g., ['Home & Kitchen', 'Artwork']
    |    |    |-- element: string, category, e.g., 'Home & Kitchen'
    |-- title: string, title of product, e.g., 'Intelligent Design Cotton Canvas'
    |-- price: float, price of product, e.g., 27.9
    |-- related: map, related information, e.g., {'also_viewed': ['B00I8HW0UK']}
    |    |-- key: string, the attribute name of the information, e.g., 'also_viewed'
    |    |-- value: array, array of product ids, e.g., ['B00I8HW0UK']
    |    |    |-- element: string product id , e.g., 'B00I8HW0UK'
2. product_processed
    |-- asin: string, same as above
    |-- title: string, title column after imputation, e.g., 'Intelligent Design Cotton Canvas'
    |-- category: string, category column after extraction, e.g., 'Home & Kitchen'
```

```
3. review
    |-- asin: string, same as above
    |-- reviewerID: string, the reviewer id, e.g., 'A1MIP8H7G33SHC'
    |-- overall: float, the rating associated with the review, e.g., 5.0
```

The `review` table will be useful for extracting the rating information for each product in Task 1. We will be working primarily with `product` table throughout Task 1-4. `product_processed` is used for Task 5-6.

All the datasets required for this assignment can be found in S3 Link[1]. We will be reading from this S3 bucket directly. You do not need to download any of them.

4 Tasks

You will be asked to complete six tasks in total. In each task you will need to implement a function `task_i()`. The function signatures and return types are fixed and provided to you in the dev-kit. Each function will take in several inputs and conduct the desired transformations. At the end of each task, you will be asked to extract several statistical properties (mean, variance, etc.) from the transformed data. You will need to programmatically put these properties in a python dictionary named `res`, the schema of which is also given.

Each of the tasks will be tested in unit. It means each function you write will be tested in isolation from the rest. We will award partial points even if some tasks failed.

All contents in this section are also contained in the jupyter notebook file provided with the dev-kit.

4.1 Conventions

These rules apply to all the tasks.

4.1.1 Results format

Each task comes with a pre-defined schema for the output results. The result must be stored as python native dictionary and must contain all the keys and nested structures.

For example the following schema:

```
res
 | -- single_value: int -- an integer number
 | -- list_of_values: list -- a list of values
     | -- element: float -- a float number
```

A desired python code snippet to compose up the dictionary would be similar to:

```
...
data = ... # Your transformed data
res = {
    'single_value': None,
    'list_of_values': [None]
} # Skeleton given for the result
res['single_value'] = data.some_op()
res['list_of_values'] = [data.some_op(), data.some_op(), data.some_op()]
...
```

4.1.2 Dealing with `null` values

The input tables contain `null` values and dangling references. You do not to deal with dangling reference unless instructed. For `null` values we will follow the common practice in SQL world. Unless instructed otherwise, you need to ignore all the `null` value entries when calculating statistics such as count, mean and variance. Of course, do not ignore `null` when you are explicitly asked to count the number of `null` entries.

[1] <s3://dsc102-pa2-public/dataset>

4.2 Task1: mean and count of ratings

First you will aggregate and extract some information from the user review table. We want to know for each product, what are the mean rating and the number of ratings it received.

1. For each product ID `asin` in `product_data`, fetch the mean value of ratings. The ratings are stored in the column `overall` of `review_data`, with product ID referencing to the former table. Store the mean value in a new column named `meanRating` in table `product_data`.
2. Similarly, put the count of ratings in a new column named `countRating`.
3. You need to conduct the above operations, then extract some statistics out of the generated columns. You need to put the statistics in a python dictionary named `res`. The description and schema of it are as follows:

```
res
 | -- count_total: int -- count of total rows of the entire table after your operations
 | -- mean_meanRating: float -- mean value of column meanRating
 | -- variance_meanRating: float -- variance of meanRating
 | -- numNulls_meanRating: int -- count of null-value entries of meanRating
 | -- mean_countRating: float -- mean value of countRating
 | -- variance_countRating: float -- variance of countRating
 | -- numNulls_countRating: int -- count of null-value entries of countRating
```

   If for a product ID, there is not a single reference in `review`, meaning it was never reviewed, you should put `null` in both `meanRating` and `countRating`.

4.3 Task 2: flattening `categories` and `salesRank`

Implement a function `task_2()` to conduct the following operations:

1. For the `product` table, each item in column `categories` contains an array of arrays of hierarchical catetories. The schema is `ArrayType(ArrayType(StringType))`. We are only going to use the most general category, which is the first element of the nested array: `array[0][0]`. Create a new column named as `category`. And for each row, put the `array[0][0]` of column `categories` in `category`. You should skip those `null` entries in `categories` and put a `null` also in `categories`. Also put a `null` if `categories` value is not `null`, but the array is empty.
2. On the other hand, each value in column `salesRank` is a dictionary with a single `(category, rank)` pair. Your task is to retrieve this key-value pair and put them in two columns, respectively. Put the category in a new column named `bestSalesCategory` and the rank in `bestSalesRank`. You should put `null` in these new columns if the original entry in `salesRank` was `null`. Note this `bestSalesCategory` may or may not be identical to `category`.
3. You need to conduct the above operations, then extract some statistics out of the generated columns. You need to put the statistics in a python dictionary named `res`. The description and schema of it are as follows:

```
res
 | -- count_total: int -- count of total rows of the entire table
 | -- mean_bestSalesRank: float -- mean value of *bestSalesRank*, excluding nulls
 | -- variance_bestSalesRank: float -- variance of *bestSalesRank*, excluding nulls
 | -- numNulls_category: int -- count of null-value entries of *category*
 | -- countDistinct_category: int -- count of all distinct values of *category*, excluding nulls
 | -- numNulls_bestSalesCategory: int -- count of null-value entries of *bestSalesCategory*
 | -- countDistinct_bestSalesCategory: int -- count of distinct values of *bestSalesCategory*,
                                            excluding nulls
```

Hint: use `DataFrame.withColumn()` to apply operations on column and add the result as a new column. To drop a column, use `DataFrame.drop()`, to rename one, use `DataFrame.withColumnRenamed()`.

Reference for Spark ML[2]

[2] <https://spark.apache.org/docs/latest/ml-features>

4.4 Task 3: flattening `related`

Inside the `related` column there is a map containing four keys/attributes: `also_bought`, `also_viewed`, `bought_together`, and `buy_after_viewing`. Each of them contains an array of `asin`s (Amazon product ID). We call these arrays attribute arrays. We need to flatten the schema by calculating the length of the arrays. In addition to the above, we would like to know the average prices of the products.

1. The logic for all four attributes are identical. For the sake of simplicity, you are only required to flatten the `also_viewed` attribute. Your task is to implement the following function `task_3()`.

2. For each row, you need to :
    1. First calculate the mean price of all products from the `also_viewed` attribute array.
    2. Then you need to put the mean price in a new column, the name of which is `meanPriceAlsoViewed`. When you calculate these mean values, remember to ignore the products if they do not match any record in `product`, or if they have `null` in price.
    3. Similary, put the length of that array in a new column `countAlsoViewed`. You do not need to check if the product IDs in that array are dangling references or not. Put `null` (instead of zero) in the new column, if the attribute array is `null` or empty.

3. You need to conduct the above operations, then extract some statistics out of the generated columns. You need to put the statistics in a python dictionary named `res`. The description and schema of which is as follows:

```
res
 | -- count_total: int -- number of rows of the entire processed table
 | -- mean_meanPriceAlsoViewed: float -- mean value of meanPriceAlsoViewed
 | -- variance_meanPriceAlsoViewed: float -- variance of meanPriceAlsoViewed
 | -- numNulls_meanPriceAlsoViewed: int -- count of null-value entries of meanPriceAlsoViewed
 | -- mean_countAlsoViewed: float -- mean value of countAlsoViewed
 | -- variance_countAlsoViewed: float -- variance of countAlsoViewed
 | -- numNulls_countAlsoViewed: int -- count of null-value entries of countAlsoViewed
```

4.5 Task 4: data imputation

You may have noticed that there are lots of `null` values in the table. Now your task is to impute them with other values that can be used.

Since we have already flattened the schema, we only have two types of values in our table: numerical (including integer and floating numbers) and string. Now you need to impute a numerical column `price`, as well as a string column `title`.

1. Please implement a function `task_4()`. For numerical column `price`, first cast it to `FloatType`. Then you want to impute the `null` values in the column with the mean value of all the not `null` values. Store the outputs in a new column `meanImputedPrice`.

2. Same as, but this time impute `null` values with the **median** value of all the not `null` values in column `price`. Store the outputs in a new column `medianImputedPrice`.

3. As for the `StringType` columns, we want to simply impute with a special string `'unknown'`. Please also impute empty strings `''`. Store the outputs in a new column `unknownImputedTitle`.

4. You need to conduct the above operations, then extract some statistics out of the generated columns. You need to put the statistics in a python dictionary named `res`. The description and schema are as follows:

```
res
 | -- count_total: int -- count of total rows of the entire table after above operations
 | -- mean_meanImputedPrice: float or None -- mean
 | -- variance_meanImputedPrice: float -- variance
 | -- numNulls_meanImputedPrice: int -- count of null-value entries
 | -- mean_medianImputedPrice: float or None -- mean
```

```
| -- variance_medianImputedPrice: float -- variance
| -- numNulls_medianImputedPrice: int -- count of null-value entries
| -- numUnknowns_unknownImputedTitle: float -- count of 'unknown' value entries
```

4.6 Task 5: embedding `title` with `word2vec`

This task assumes the ***`title`*** *column is already imputed with* ***`unknown`****. We will provide the imputed data table* ***`product_processed_data`****.*

In this task we want to transform `title` into a fixed-length vector by training and then applying word2vec.

1. You need to implement function `task_5()`. For each row,
   1. convert all characters in `title` to lowercase,
   2. then split `title` by whitespace (`' '`) to an array of strings, store it in a new column `titleArray`
2. Train a word2vec model out of this column `titleArray`, then for each row, transform `titleArray` into vectors. First transform every word in the array to vector, then simply averaging the vectors to obtain the vector for the title. Put the title vector in a new column named `titleVector`. Do not try to implement word2vec yourself, instead, use `M.feature.Word2Vec` and it has built-it method to do the transformation. See instructions below.
3. Use your trained word2vec model to get the 10 closest synonyms along with the similarity score (based on cosine similarity of word vectors, descending) each for three words inputed as `<word_0>`, `<word_1>`, and `<word_2>`. `M.feature.Word2Vec` also has built-in method for this task.
4. You need to conduct the above operations, then extract some statistics out of the generated columns. You need to put the statistics in a python dictionary named `res`. The description and schema is as follows:

```
res
 | -- count_total: int -- count of total rows of the entire transformed table
 | -- size_vocabulary: int -- the size of the vocabulary of your word2vec model
 | -- word_0_synonyms: list -- synonyms tuples of word_0
 |    | -- element: tuple -- tuple of format (synonym, score)
 |    |    | -- element: string -- synonym
 |    |    | -- element: float -- score
 | -- word_1_synonyms: list
 |    | -- element: tuple
 |    |    | -- element: string
 |    |    | -- element: float
 | -- word_2_synonyms: list
 |    | -- element: tuple
 |    |    | -- element: string
 |    |    | -- element: float
```

word2vec instructions:

1. Set `minCount`, the minimum number of times a token must appear to be included in the word2vec model's vocabulary to be `100`.
2. Set the dimension of output word embedding to `16`.
3. You need to set the random seed as `SEED`, this is a global variable defined to be 102.
4. Set `numPartitions` to be `4`.
5. You should keep all other settings as default.
6. `M.feature.Word2Vec` is not fully reproducible (although we have set the seed here). We are aware of the issue and your score will not be affected by its internal randomness.

Reference for word2vecs on Spark ML[3]

[3] `<https://spark.apache.org/docs/latest/ml-features.html#word2vecs>`

4.7 Task 6: one-hot encoding `category` and PCA

Assume the schema of ***`categories`*** *is already flattened and* ***`unknown`*** *imputed for the input data. We will provide you with the preprocessed table*

Now you need to one-hot encode the categorical features. Also, these categories may be correlated and as a practice, we want to run PCA on these categories.

1. Implement function `task_6()`. First one-hot encode `category` and put the output vectors in a new column `categoryOneHot`. Note you do need to ensure the dimension of generated encoding vector equals to the size of domain. For example, if we have three categories in total: `V = {'Electronics', 'Books', 'Appliances'}`. Then the encoding of 'Electronics' can be `[1, 0, 0] or [0, 1, 0] or [0, 0, 1]` but the dimension of this vector must be 3. Hint: before one-hot encoding a StringType column, you may need to first convert that column of strings to a column of numerical indices with `M.feature.StringIndexer`. Then use `M.feature.OneHotEncoderEstimator` to do the encoding with `dropLast` argument set to false.

2. Second, use `M.feature.PCA` on the one-hot-encoded column. Reduce the dimension of each one-hot vector to `15`, put the transformed vectors in a new column `categoryPCA`.

3. Column `categoryOneHot` and `categoryPCA` will be of VectorType. You do not need to worry about if the vectors are sparsely or densely represented.

4. You need to conduct the above operations, then extract some statistics out of the generated columns. You need to put the statistics in a python dictionary named `res`. The description and schema is as follows:

```
res
 | -- count_total: int -- count of total rows of the entire transformed table
 | -- meanVector_categoryOneHot: list -- mean vector of transformed one-hot-encoding vectors
 |    | -- element: float -- each element of the mean vector, from first to last dimension
 | -- meanVector_categoryPCA: list -- mean vector of PCA-transformed vectors
 |    | -- element: float
```

5 Deliverables

Code up all the tasks in the designated places in `assignment2.py`. Then rename the file to `assignment2_<your team id>.py`. For instance, if your team id is 18, then your filename would be `assignment2_18.py`. Submit this file on Canvas, only one team member needs to do so.

6 Getting started

6.1 Prerequisite

You need to download Docker on your own computer and prepare the key pairs on AWS.

6.1.1 Docker

The only dependency of this assignment is docker, which can be found in Link[4]. A container image named `yuhzhang/dsc102-pa2` will be used for all AWS-related operations.

Windows users: Your system may not fulfill the requirements for the newest Docker Desktop. In that case, please install Docker Toolbox[5] instead.

Within the container you are provided with the following utilities

```
s3-init
emr-launch
emr-list
emr-dns
emr-terminate
```

The entry point of these utilities is

[4] `<https://docs.docker.com/install>`
[5] `<https://docs.docker.com/toolbox/toolbox_install_windows>`

```
docker run --env-file <path> yuhzhang/dsc102-pa2 <utility>
```

`<path>` is the path to your credential file, see below for instructions. `<utility>` is any utility listed above. The AWS CLI tool is also avaliable as `aws2`. For example, to list your s3 bucket, you can run:

```
docker run --env-file <path> yuhzhang/dsc102-pa2 aws2 s3 ls
```

6.1.2 Prepare key pairs on AWS console

If you have not done so. Go to your AWS console[6] and click key pairs. Follow the instructions to create or upload a public key to AWS. Note down the name of the key as displayed on your AWS console.

6.2 Store AWS credentials

Create a new text file named `credentials.list` under any directory and put the following content in it:

```
PID=...
AWS_ACCESS_KEY_ID=...
AWS_SECRET_ACCESS_KEY=...
AWS_SESSION_TOKEN=...
```

Fill in the blanks (placeholded by `...`) with corresponding values. `PID` is your UCSD pid (e.g., a13230999, with 'a' in lower case). You can find the rest three AWS credentials from Link[7]. Note these credentials are only temporary. You will need to update this file if the token expires. To obtain a new token, simple revisit the link.

6.3 Initialize assignment-related S3 buckets

Use the following command to initialize the S3 buckets needed for this assignment:

```
docker run --env-file path/to/credentials.list yuhzhang/dsc102-pa2 s3-init
```

This command will setup a EMR log bucket named `<your pid>-emr-logs`, and a bucket storing your scripts named `<your pid>-pa2`. It will also copy the assignment dev-kit to the latter bucket.

6.4 Launch an EMR cluster

1. Use `emr-launch` utility built in the docker image `yuhzhang/dsc102-pa2`:

   ```
   docker run --env-file path/to/credentials.list yuhzhang/dsc102-pa2 emr-launch -k <key name>
   ```

   The above utility will spawn a EMR cluster named `<pid>-emr-cluster`. Available flags/arguments are:

   ```
   -k <key name>, the name of your secrets to access the cluster via ssh
   -b, optional, if set, your CORE instances will be run with Spot pricing
   -n <number of workers>, optional, the number of workers you want to have, default: 4
   -d, optional, if set, use the deployment hardware m5.xlarge, otherwise use m4.large
   -t, optional, if set, setup Theia IDE on master node port 3000
   -f, optional, if set, force starting the second cluster, as by default only one cluster is permitted
   ```

2. Example usage

   Spawn a development cluster with 4 workers and Spot pricing:

   ```
   docker run --env-file path/to/credentials.list yuhzhang/dsc102-pa2 emr-launch -k <key name>
       -b -n 4
   ```

   Spawn a deployment cluster:

   ```
   docker run --env-file path/to/credentials.list yuhzhang/dsc102-pa2 emr-launch -k <key name>
        -n 4
   ```

   This deployment environment is **fixed** and will be used to evaluate all of your submissions.

[6] <https://ets-apps.ucsd.edu/dsc102-custom-aws/>
[7] <https://ets-apps.ucsd.edu/dsc102-custom-aws/?mode=env>

6.5 Access your cluster and the assignment

1. Use the following command to list the cluster IDs:

```
docker run --env-file path/to/credentials.list yuhzhang/dsc102-pa2 emr-list
```

2. Then query the DNS name of cluster's master node using:

```
docker run --env-file path/to/credentials.list yuhzhang/dsc102-pa2 emr-dns <cluster ID>
```

   This will return the DNS name such as:

   `ec2-###-##-##-###.compute-1.amazonaws.com`

3. (Optional) You can now try to SSH into your master node via:

```
ssh -i path/to/key hadoop@ec2-###-##-##-###.compute-1.amazonaws.com
```

   Log out the ssh session before proceeding to the next steps.

4. In your browser, access

   `ec2-###-##-##-###.compute-1.amazonaws.com:8888`

   this will direct you to the jupyter notebook running on the master node. The password would be you pid. The working directory of this jupyter notebook is S3 bucket `<your pid>-pa2` mounted. So all your modifications to the assignment files will be reflected to that bucket.

5. (Optional) and if you used `-t` flag when launching the cluster, you can access Theia IDE running on

   `ec2-###-##-##-###.compute-1.amazonaws.com:3000`

6. In Jupyter notebook, rename `assignment2.ipynb` to `assignment2_<your pid>.ipynb` and continue the assignment by following the instructions written in the notebook.

6.6 Testing and submission

You will **not** submit the notebook. Instead, you need to put your implementations of `task_1` to `task_6`, along with all the dependencies you imported and helper functions you defined, in the file co-located with the notebook: `assignment2.py`.

If you are collaborating in team, please combine your work into one single file. Only **one** person needs to submit the final file. Do **not** modify the filename yet.

6.6.1 Test your file

Before submitting the file, you need to make sure your script runs under the deployment environment, otherwise you may lose points.

1. If your cluster is not in deployment mode, meaning if you don't have 4 workers, terminate your current cluster (see instructions below), then launch a deployment cluster via

```
docker run --env-file path/to/credentials.list yuhzhang/dsc102-pa2 emr-launch -k <key name>
    -n 4
```

2. SSH into the master node by

```
ssh -i path/to/key hadoop@ec2-###-##-##-###.compute-1.amazonaws.com
```

3. Go to your root directory of scripts

```
cd /mnt/<your pid>-pa2/src
```

4. Run PA2 with the following command, do not modify anything except `<your pid>`:

```
spark-submit \
--py-files utilities.py,assignment2.py \
--files log4j-spark.properties \
--master yarn \
--deploy-mode client \
--conf spark.memory.fraction=0.8 \
--conf spark.dynamicAllocation.enabled=false \
 --conf spark.sql.crossJoin.enabled=true \
 --driver-java-options "-Dlog4j.configuration=file:log4j-spark.properties" \
 --conf "spark.executor.extraJavaOptions=-Dlog4j.configuration=file:log4j-spark.properties"
    \
 pa2_main.py --pid <your pid>
```

Make sure your script can execute and try to pass as many tests as you can.

6.6.2 Submit your file

Go to your AWS console, navigate to S3 buckets and find the bucket named `<your pid>-pa2`. Download the `assignment2.py` file to your own machine.

Then rename the file to `assignment2_<your team id>.py`. For instance, if your team id is 18, then your filename would be `assignment2_18.py`.

Upload this file to canvas, only one of the team members needs to do so.

6.7 Terminate your cluster

Don't forget to terminate the cluster when you are done:

```
docker run --env-file path/to/credentials.list yuhzhang/dsc102-pa2 emr-terminate <cluster id>
```