Midterm, Spring 2020

Due April 15, 11:55PM, via NYU Classes

** Submit a single zip file, with your netId and 'midterm' as name, with a subfolder for each question, and code and/or jupyter notebook in their respective folders.

1. Hadoop Map Reduce – Sampling a dataset. 30 points

Data: cookbook.zip

Submit: zip file code file(s)

Imagine you're working with a terabyte-scale dataset and you have a MapReduce application you want to test with that dataset. Running your MapReduce application against the dataset may take hours, and constantly iterating with code refinements and rerunning against it isn't an optimal workflow.

To solve this problem you look to sampling, which is a statistical methodology for extracting a relevant subset of a population. In the context of MapReduce, sampling provides an opportunity to work with large datasets without the overhead of having to wait for the entire dataset to be read and processed.

TO-DO:

- Write a map/reduce program that implements a random sampler.
- The program that you'll write should be configured, via arguments, with the number of samples that should be extracted from the input.
- The input test data for this problem is in the cookbook.zip file in NYU Classes. Specifically, the files matching cookbook_text/*.txt pattern.
- The sampling method ***must*** be the reservoir sampling algorithm: http://en.wikipedia.org/wiki/Reservoir_sampling.

2. Spark Dataframes
27 points

Data: data_Q2_SP20.zip

Submit: python notebook

(3 points each) Spark Scala, Spark Java or PySpark

a) Create a single dataframe from all CSV files in the zip, with header information
b) Show the dataframe columns
c) Show the first 20 rows, sorted by (capacity descending, model ascending)
d) Count the total number of rows
e) Count the total number of rows, grouped by capacity
f) Get the dataframe summary statistics
g) Select the following columns: date, model, capacity
h) Select the number of distinct models
i) Calculate the pairwise frequency of this two columns (e.g. crosstab): capacity, smart_1_normalized
j) Find the mean value of column capacity

3. Spark Anomaly detection – Hard Drive Failures 60 points

Data: **docs_Q3_SP20.zip**, **data_Q2_SP20.zip**
https://www.backblaze.com/blog/backblaze-hard-drive-stats-q1-2019/

Reference: https://www.backblaze.com/b2/hard-drive-test-data.html

INTRODUCTION

Anomaly detection is a technique used to identify unusual patterns that do not conform to expected behavior, called outliers.

Anomalies can be broadly categorized as:

1. **Point anomalies:** A single instance of data is anomalous if it's too far off from the rest. *Business use case:* Detecting credit card fraud based on "amount spent."
2. **Contextual anomalies:** The abnormality is context specific. This type of anomaly is common in time-series data. *Business use case:* Spending $100 on food every day during the holiday season is normal, but may be odd otherwise.
3. **Collective anomalies:** A set of data instances collectively helps in detecting anomalies. *Business use case:* Someone is trying to copy data form a remote machine to a local host unexpectedly, an anomaly that would be flagged as a potential cyber attack.

TO-DO

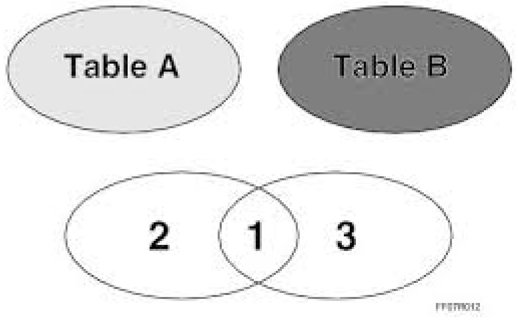
In Spark ML (**not pandas**): given the hard drive logs for 2019 Q1, implement a *point anomaly* detector for:

a) **Annualized Failure Rate (by model)**
b) **Normalized Read Error Rate, SMART attribute 1.**

- For generating training labels, use a) 2% b) 100
- For an explanation of hard drive SMART attributes, refer to: https://en.wikipedia.org/wiki/S.M.A.R.T.#Known_ATA_S.M.A.R.T._attributes
- For an explanation of scripts required for these computations, refer to **docs_Q1_2019.zip.**
- **Anomaly Detection Reference: https://www.datascience.com/blog/python-anomaly-detection**
- Grading: ETL/Data formatting 30 points, ML train/predict/report 30 points

4. Spark Bloom Filters and Broadcast Joins
50 points

Suppose you are interested in records on one dataset, Table A, based on values of another dataset, Table B. Generally, an inner join is used as a form of filtering.



Consider, for example, if Table A has 100's of millions of rows, while Table B has only a few thousands.

In cases like this, you might want to ***avoid the shuffle*** that the join operation introduces, especially if the dataset you want to use for filtering is ***significantly smaller*** than the main dataset on which you will perform your further computation.

Definition: Broadcast Join

In a broadcast join, you send the ***entire*** smaller dataset, Table B in our case, to **each** node/worker in our cluster. A join in this manner is called a broadcast join.

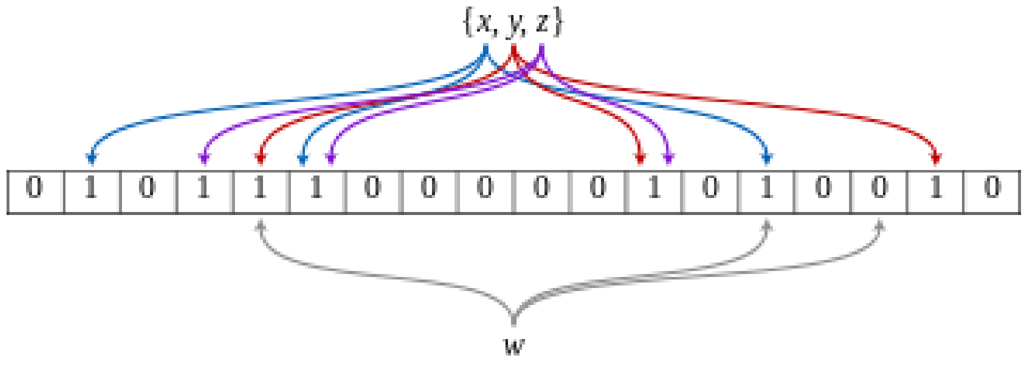
Spark Broadcast Variables:
https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html#broadcast-variables

"Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used, for example, to give every node a copy of a large input dataset in an efficient manner."

Definition: Bloom Filter
https://en.wikipedia.org/wiki/Bloom_filter

Bloom filters are efficient probabilistic data structures constructed of a set of values to be used for membership tests. It can tell you if an arbitrary element being tested **might** be in the set, or **definitely not** in the set, that is false positives are allowed, but false negatives are not.

The data structure is a bit array, onto which elements are mapped using a hash function. The mapping basically sets some bits to 1 leaving the rest as 0's. The size of the bit array is determined by how much false positives you are willing to tolerate, so most implementations accept an FPR param in the constructor of the data structure (typical value is 1%).



An example of a Bloom filter, representing the set {*x*, *y*, *z*}. The colored arrows show the positions in the bit array that each set element is mapped to. The element *w* is not in the set {*x*, *y*, *z*}, because it hashes to one bit-array position containing 0. For this figure, *m* = 18 and *k* = 3.

TO-DO

Implement a broadcast join using Bloom filters.

Using broadcast join and a bloom filter, filter all rows in Table A for only those containing 'model' in Table B. The common key is the '**model'** column.

Note:

Data Source: data_Q2.SP20.zip

Instructions:

Table A: all files from 2019-01-01.csv through 2019-03-30.csv

Table B: 2019-03-31.csv

Reference: https://www.duedil.com/engineering/efficient-broadcast-joins-using-bloom-filters

5. EXTRA CREDIT: Duplicate Detection/Minhash – 50 points

Implement the Locality Minhash/LSH algorithm discussed in class, using Spark (Scala, Java, or Python). Your code must be runnable, and in a Zeppelin or Jupyter notebook format.

Here are some references as well:
http://mccormickml.com/2015/06/12/minhash-tutorial-with-python-code/
https://mattilyra.github.io/2017/05/23/document-deduplication-with-lsh.html

The input to your duplicate detector will be the text documents from cookbook_text.zip.