CS450 Assignment 3

Katya Griffiths-Julien, Jerry Lau

There were no provided trials and test income CSV files in the moodle page. After emailing the professor, he explained to reuse the files from Assignment 1. Thus, the trial file has 1 000 numbers for input and the test has 10 000 000 numbers. Both files do not have any headers.

Task 1

In pyspark using the distinct function returns a new DataFrame with only the distinct rows from the original DataFrame. Then we apply count() so that we get a value of all the distinct rows and not the entire DataFrame. This value is then the number of distinct items in the CSV.

Task 2

There is a function for median() in pyspark however that cannot be used since we do not have column headers in our CSV file. Then we could also approximate the quantile but the instruction specify that it was not allowed. Therefore, we had use some additional manipulation to get the median since we were restricted to using specific shufflable transformations.

Similarly to task 3, the DataFrame is first grouped into their values and then counted the total number of values. Then, they are then grouped by ascending values. Those values are then placed into a numpy array which is then raveled so that the array traversal is made simpler. Then we use a for loop to traverse the array and get the index at which the median. We know how big the array is so we only need to get to the half way. Finally we need to do add 1 to the index since it starts at 0 while the smallest CSV value is 1 so an offset is required.

Task 3

For this task, we set the DataFrame to group by the value of the only column in the CSV. Then we count the values that are grouped. Afterwards, we call order by the counted values in reverse such that the largest value is in the front. Then we take the 1st value from the DataFrame which is the value that is most frequently seen in the CSV.

Task 4

In this task, create a new column in the DataFrame which does the log10 and floor of each value in the parsed CSV file. Then the values are once again grouped and then counted to see which values show up more often. Lastly, we sort the DataFrame such that the rounded values from the log10 operation are in ascending order.

Conclusion

From simple test using the NumPy library instead of using PySpark, it seems that the computational overhead for PySpark is much higher than simple NumPy. Using the trial dataset, to calculate or find the distinct values, mode and median in NumPy takes about 0.169s while PySpark requires 15.757s. This is an increase in computation time of 2 factors of ten.

The use of PySpark for these statistical tasks does not provide us with any gains than the use of NumPy or any other statistical libraries. In fact, it actually reduces the overall performance. This is mainly because PySpark is not designed for statistical analysis, rather it is more for data processing in SQL like datasets or relational algebra type queries. This is clear to see in the notebook file which has a list of the various joins available in PySpark. These joins are not useful in our situation since the dataset is only 1 file and only had 1 column. As such, it is difficult to leverage the entirety of the PySpark library and to make use of its strengths.

Rather, we are handicapping what PySpark is not good at and the performance results against NumPy leave much to be desired. Therefore, using the estimates and the proposed counting sort type of function from Assignment 1 with a limited dataset performs better than PySpark. This is both in terms of time and memory requirements.