Code Review: a5-navya-shabbir-yu

Bhanu, Joyal, Jiangtao

October 22, 2017

Contents

Details

Review

a5-navya-shabbir-yu

Reviewer

Bhanu, Joyal, Jiangtao

Codebase

https://github.ccs.neu.edu/pdpmr-f17/a5-navya-shabbir-yu

Report Review

- Report is well structured and explains each component well.

Code Review

- Should have completion check on Job Result.

  ```
  File: a5-navya-shabbir-yu/src/main/java/org/neu/pdpmr/tasks/Main.java
  96:         new ExtractData(this.inDir, this.outDir, trainDatDir, qryYearDataDir, this.maxTrainYear
  ```

  At L96 you can check whether `ExtractData` is completed or not. Any exception other than `IOException, InterruptedException, ClassNotFoundException` will be consumed at job level and there is no mechanism to know the job status other than the return value of `job.waitForCompletion`. If `job.waitForCompletion == -1`, you should halt any further execution. With current code, even if `job.waitForCompletion == -1` we still move on to the next task.

- Invalid Data Formats for Time Fields

```
File: a5-navya-shabbir-yu/src/main/java/org/neu/pdpmr/tasks/mappers/DataExtractingMapper.java
66:         float timeZone          = crsArrTime - crsDeptTime - crsElapsedTime;
67:         float cancelledCheck    = timeZone - actElapsedTime;
```

`crsArrTime, crsDeptTime, crsElapsedTime` considered as float while they are actually in time format `hhmm`

- Avoid Consecutive Casting

  ```
  File: a5-navya-shabbir-yu/src/main/java/org/neu/pdpmr/tasks/mappers/DataExtractingMapper.java
  46:             context.write(new IntWritable((int) ((float) f.get("YEAR"))), datin);
  47:         } catch (Exception e) {
  ```

  `Year` is a `int`, but was taken as `float`. As a result authors had to cast from `Object` to `float` to `int`. This pattern is seen everywhere in the code. You can set `year` to `int` in your parser, as `year` can never be `float`

- Authors are parsing CSV records twice in both `DataExtractingMapper` and `DataExtractingReducer`. It is unnecessary compute. You can parse once in `DataExtractingMapper` and extract only meaningful fields. These fields can be converted into custom writable values and emitted from mapper. Thus in the reducer you don't have to parse CSV texts again.

- You can get the below year directly from the `key`.

  ```
  File: a5-navya-shabbir-yu/src/main/java/org/neu/pdpmr/tasks/reducers/DataExtractingReducer.java
    63:               // year in the record
    64:               int recordYear = (int) ((float) f.get("YEAR"));
    65:
  ```

- DataExtractingReducer.LOC101: `min` is already `1`, so `max - min + 1` would always be `max`

  ```
  File: a5-navya-shabbir-yu/src/main/java/org/neu/pdpmr/tasks/reducers/DataExtractingReducer.java
  098:              Random rand = new Random(System.currentTimeMillis());
  099:              int max =10000;
  100:              int min =1;
  101:              int res =rand.nextInt(max - min + 1) + min;
  ```

- Instead of manually merging `MultiPartOutput` you can use `LazyOutputFormat.setOutputFormatClass(job, TextOutputFormat.class)` to get merged results from reducer. (Assuming you need to merge only the multipart output.)

- Unecessary Query serialization/serialization

  Reading queries in `Main.java` -> converting it to Query Collection -> converting it to JSON string -> setting this string into job config -> converting this string to Query Collection in Job2

  This seems to be a very complicated process with unnecessary conversions and parsing. You can simply read query file in job2.

- The authors have used counters, but there is no explanation of what these counters are doing. Cannot understand just by reading the code.

  ```
  File: a5-navya-shabbir-yu/src/main/java/org/neu/pdpmr/tasks/mappers/TwoHopFlightsMapper.java
  118:        // flightDate >= journeyDateMs && flightDate <= journeyDateMs + X_DAYS
  119:        context.getCounter("MyMap", "s1").increment(1);
  120:        if (!(flightDateMs >= q.dateMs)) return;
  121:        context.getCounter("MyMap", "s2").increment(1);
  122:        if (!(flightDateMs <= q.dateMs + THREE_DAYS)) return;
  123:        context.getCounter("MyMap", "s3").increment(1);
  124:
  ```

- The join operation to generate two hop flights is very complex and missing documentation. Cannot comprehend just by reading the code.

Execution

- The implementation has memory issues, and the code does not run completely as mentioned in the report.
- `make all` is missing `exec` target to run hadoop job.

Final Thoughts

- Cannot verify the accuracy as the results are not generated.
- Code quality is not very good and hence making it very difficult to read.
- Documentation about joins and code structure would be very helpful.
- Random Forest is generally a memory intensive algorithm. You can try simple models such as `average by year`, or linear classifications to optimize memory usage.
- You can avoid using complex data structures such as `TypeToken` and `GSON` serializations.