Introduction to Data Science WS 24/25

Report Assignment Part 2

Group 043*:*
Yiğit Kaan Yıldız (464480)
Zhongjian Rong (464649)
Ali Berger (383691)

Statement on the usage of LLMs

⟨If you make use of generative AI in the form of LLMs, state in which tasks you used them for which purpose. You may also further argue why the specific usage does not detract from your understanding of a particular task. ⟩

Q1 Distributed Data Processing

Q1.1 a

```python
import mrjob
from collections.abc import Iterator, Iterable
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.protocol import TextValueProtocol, PickleValueProtocol


class BestMovies(MRJob):
    # this pre-processing mapper is for convenience
    # it unpacks the string value into the logical signature of the file
    def map_pre(self, _: None, line: str) -> Iterator[tuple[int, tuple[str, str, float, str, float]]]:
        i, user_id, movie_id, rating, timestamp, rating_normalized = line.split(',')
        yield i, (user_id, movie_id, float(rating), timestamp, float(rating_normalized))

    def map_1(self, _: int, line: tuple[str, str, float, str, float]):
        user_id, movie_id, rating, timestamp, rating_normalized = line
        yield movie_id, (1, rating)

    def reduce_1(self, movie_id, values):
            total_reviews = 0
            total_ratings = 0
            for (reviews, rating) in values:
                    total_reviews += reviews
                    total_ratings += rating
            avg = total_ratings/total_reviews
            if avg >= 4 and total_reviews >= 10:
                    yield movie_id, avg

    def map_2(self, movie_id, avg):
            yield None, (avg, movie_id)

    def reduce_2(self, _, values):
            topten = []
            for p in values:
                    topten.append(p)
                    topten.sort()
                    topten = topten[-8:]

            for (avg, movie_id) in topten:
                    yield None, (movie_id, avg)


    # this post-processing mapper is for convenience
    # the output from the last reducer step is simply written as text into a file, ignoring the key
    def map_post(self, _, pair: tuple[str, float]):
        yield None, ','.join(map(str, pair))

    # the output from the last reducer step is simply written as text into a file, ignoring the key
    OUTPUT_PROTOCOL = TextValueProtocol

    # this can be treated as boilerplate
    def steps(self):
        return [MRStep(mapper=self.map_pre),
                MRStep(mapper=self.map_1, reducer=self.reduce_1),
                MRStep(mapper=self.map_2, reducer=self.reduce_2),
                MRStep(mapper=self.map_post)]


if __name__ == '__main__':
    BestMovies.run()
```

movie name	average rating
Ran (1985)	4.433333333333334
Secrets & Lies (1996)	4.590909090909091
Guess Who's Coming to Dinner (1967)	4.545454545454546
His Girl Friday (1940)	4.392857142857143
Paths of Glory (1957)	4.541666666666667

There are 8 records in the csv file, but the latex csvsimple package doesn't work well with csv files with special characters such as ",", and it only shows 5 records here. The full records can be seen in the top8.csv file.

Q1.2 b

```
import mrjob
from collections.abc import Iterator, Iterable
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.protocol import TextValueProtocol


class Haters(MRJob):


    def mapper(self, _, line: str):
            i, user_id, movie_id, rating, timestamp, rating_normalized = line.split(',')
            if float(rating) < 2:
                    yield user_id, 1

    def combiner(self, key, values):
            yield key, sum(values)


    def reducer(self, key, values):
            reviews = sum(values)
            if reviews >= 50:
                    yield None, key


if __name__ == '__main__':
    Haters.run()
```

Q1.3 c

Before handing the outputs of mappers to reducers, the combiners intercept the pairs and combine the counts of reviews with the same key, though in the sense of grouping in subsets. This reduces the communication overhead that is sent to the reducers. The best-case scenario is when the combiners get the similar workloads, so that with the parallel processing, no combiners is processing too much data.